RADC-TR-90-404, Vol II (of 18)
Final Technical Report
December 1990

DISCUSSING, USING, AND RECOGNIZING PLANS

Northeast Artificial Intelligence Consortium (NAIC)

Stuart C. Shapiro and Beverly Woolf

This effort was funded partially by the Laboratory Director's fund.

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Services (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-90-404, Volume II (of 18) has been reviewed and is approved for publication.

APPROVED: Sharon M. Walter

SHARON M. WALTER
Project Engineer

APPROVED: Raymond P. Urtz Jr.

RAYMOND P. URTZ, JR.
Technical Director
Directorate of Command & Control

FOR THE COMMANDER: Ronald Raposo

RONALD RAPOSO
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COES) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1990	Final Sep 84 - Dec 89

4. TITLE AND SUBTITLE
DISCUSSING, USING, AND RECOGNIZING PLANS

5. FUNDING NUMBERS
C - F30602-85-C-0008
PE - 62702F
PR - 5581
TA - 27
WU - 13
(See reverse)

6. AUTHOR(S)
Stuart C. Shapiro and Beverly Woolf

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Northeast Artificial Intelligence Consortium (NAIC)
Science & Technology Center, Rm 2-296
111 College Place, Syracuse University
Syracuse NY 13244-4100

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Air Development Center (COES)
Griffiss AFB NY 13441-5700

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RADC-TR-90-404, Vol II (of 18)

11. SUPPLEMENTARY NOTES (See reverse)
RADC Project Engineer: Sharon Walter/COES/(315) 330-3577

This effort was funded partially by the Laboratory Director's fund.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The Northeast Artificial Intelligence Consortium (NAIC) was created by the Air Force Systems Command, Rome Air Development Center, and the Office of Scientific Research. Its purpose was to conduct pertinent research in artificial intelligence and to perform activities ancillary to this research. This report describes progress during the existence of the NAIC on the technical research tasks undertaken at the member universities. The topics covered in general are: versatile expert system for equipment maintenance, distributed AI for communications system control, automatic photointerpretation, time-oriented problem solving, speech understanding systems, knowledge base maintenance, hardware architectures for very large systems, knowledge-based reasoning and planning, and a knowledge acquisition, assistance, and explanation system.

The specific topic for this volume is the recognition of plans expressed in natural language, followed by their discussion and use.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Artificial Intelligence, User Interfaces, Hypothetical Reasoning, Planning, Natural Language, Plan Recognition	138
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Block 5 (Cont'd) Funding Numbers

PE - 62702F	PE - 61102F	PE - 61102F	PE - 33126F	PE - 61101F
PR - 5581	PR - 2304	PR - 2304	PR - 2155	PR - LDFP
TA - 27	TA - J5	TA - J5	TA - 02	TA - 27
WU - 23	WU - 01	WU - 15	WU - 10	WU - 01

Block 11 (Cont'd)

This effort was performed as a subcontract by the State University of Massachusetts at Amherst to Syracuse University, Office of Sponsored Programs.

2 DISCUSSING, USING, AND RECOGNIZING PLANS

Report submitted by:

Stuart C. Shapiro (UB)
Beverly Woolf (UMASS)
Syed S. Ali (UB)
Scott Anderson (UMASS)
David R. Forster (UMASS)
Juergen Haas (UB)
Deepak Kumar (UB)
James Pustejovsky (Brandeis U.)
Penelope Sibun (UMASS)

Department of Computer Science
SUNY at Buffalo(UB)
226 Bell Hall
Buffalo, NY 14260

Computer and Information Science
Graduate Research Center
University of Massachusetts(UMASS)
Amherst, MA 01003

A-1

TABLE OF CONTENTS

2 DISCUSSING, USING AND RECOGNIZING PLANS

2.1 INTRODUCTION

This project, also known as the Natural Language Planning project, has been a joint project of a research group at SUNY at Buffalo (UB), led by Dr. Stuart C. Shapiro, and a research group at the University of Massachusetts at Amherst (UMass) led by Dr. Beverly Woolf. The project was devoted to the investigation of a knowledge representation design compatible with the intensional knowledge representation theory previously developed by Dr. Shapiro and his co-workers and capable of providing a natural language interacting system with the ability to discuss, use, and recognize plans. The UB group was responsible for: the development, improvement, and maintenance of the knowledge representation, reasoning, and natural language processing software to be used in the project; for developing a representation of plans and associated concepts; and for developing basic techniques for discussing plans in English, and for plan recognition. The UMass group was responsible for analyzing the chosen domains, principly the domain of tutoring interactions, for testing the developments of the Buffalo group by trying to use plans in comprehending natural language paragraphs, and for suggesting changes to the representation of plans. With the support of the NAIC and of Texas Instruments, both groups used TI Explorers to do their work.

2.1.1 Objectives

The objectives of this project were to:

1. design a representation for plans and rules for reasoning about plans within an established knowledge representation/reasoning (KRR) system; enhance the KRR system so that it can act according to such plans;

2. write a grammar to direct an established natural language processing (NLP) system to analyze English sentences about plans and represent the semantic/conceptual content of the sentences in the representation designed for objective (1); the resulting NLP system should be able to: accept sentences describing plans, and add the plans to its "plan library"; answer questions about the plans in its plan library; accept sentences describing the actions of others, and recognize when those actions constitute the carrying out of a plan in its plan library.

The KRR system used was SNePS [24], and the NLP system to be modified for this purpose was CASSIE [30]. The UB group was responsible for enhancing SNePS/CASSIE according to the objectives listed above, using the Blocksworld as an initial development/testing domain. The UMass group was responsible for testing the enhanced system in the specific domain of tutoring.

2.1.2 Overview

This report describes in detail how the objectives outlined above were met. We discuss the design, implementation, and use of representations for plans to model a cognitive agent whose behavior is driven by its beliefs, desires, and intentions. We will give the motivations underlying our representations for plans, goals, acts, actions, pre-conditions and post-conditions. These representations are designed to satisfy constraints posed by the issues involved in fulfilling the tasks mentioned above (natural language understanding, belief representation, planning and problem solving, plan recognition, and text generation).

We built a system that reads a paragraph about a situation in some domain and then demonstrates its understanding of the paragraph by answering questions about the situation. Two domains were of primary interest: rocket launching, because of its potential usefulness in machine-scanning reports of rocket launches; and tutoring, because its complexity would drive the research and demand greater competence of the implementation. The first domain was eventually dropped because of the impossibility of getting realistic but "laundered" data, though we feel confident that our system could be made to handle the rocket launch domain in a relatively short time.

The tutoring domain fulfilled its purpose, because we had to stretch our understanding of planning and goal-driven behavior, and we were confronted with texts that required new techniques of Natural Language Processing (NLP). This report reviews our research into Tutoring as Planning (Section 2.11), and our NLP techniques for Understanding Situations (Section 2.12). Next, it gives an overview of our implemented program, which we call TOM, for Tutoring Observation Module. Section 2.13 also covers our implementation of the grammar, the rules in the knowledge base, and the focussing algorithm.

SNePS, the knowledge representation/reasoning system used for this project, and its associated Generalized Augmented Transition Network (GATN) grammar interpreter/compiler were implemented in Franz Lisp when this project started. Significant steps had been taken to redesign and reimplement the software, now to be called SNePS-2, in Common Lisp. This was crucial for cooperation between the UB and UMass groups, since both were to use Common Lisp on TI Explorer Lisp Machines. However, when the project started, much work was left to be done.

During the course of the project, the UB group implemented enough of SNePS-2 to be useful (The inference package is still not complete, but is already a superset of Horn Clause logic.), redesigned and reimplemented major portions of it, wrote a usable manual, and, with the help of a related group at the Technical University of Lisbon, included belief revision as a standard feature of SNePS, now called SNePS-2.1.

The UMass group received a copy of SNePS-2.0 early in the course of the project and contributed to its debugging and development. However, mid-way through the project, they ceased to keep up with the UB revisions, and maintained their own copy of the system in order to more expeditiously pursue their major research objectives.

We then discuss our experiences with using SNePS at a new site, our laboratory at the University of Massachusetts (Section 2.14). Finally, we review our research conclusions and point to directions for future research.

Overview of the system

Our work is proceeding by implementing, experimenting with, and revising a system called SNACTor. SNACTor begins with an empty knowledge-base. In the role of informant, we interact with SNACTor using English sentences about the domain, instructing SNACTor about the various actions that it can perform, and how to solve problems in that domain. The input sentences are analyzed using a domain-specific grammar, the results of which are new beliefs in the knowledge-base. A natural language generation grammar takes the new beliefs and expresses them back in English to indicate SNACTor's understanding to the informant. Requests to perform some action are sent to an acting executive that may then generate and execute a plan to fulfill the request. The informant may also ask questions about plans and the way the system would solve various problems.

A Generalized ATN grammar [25] is used for analyzing input sentences and for generating English responses. SNACTor currently operates in a Blocksworld domain.

2.2 MOTIVATIONS UNDERLYING OUR REPRESENTATIONS

Our goals were to design and implement representations for plans to model a rational cognitive agent whose behavior is driven by its beliefs, desires, and intentions. We now give the motivations underlying our representations of plans, goals, acts, actions, preconditions and post-conditions. As mentioned before these representations are designed to satisfy constraints posed by issues in natural language understanding, belief representation, planning and problem solving, plan recognition, and text generation. A preliminary version of this work appears as an extended abstract in [27]. Since then, there have been changes in the design of our representations and planning techniques used [28]. A detailed account of the syntax, and semantics of our earlier representations can be found in [12].

2.2.1 Motivation for intensional representations of plans

Georgeff (1987) mentions the importance of "considering planning systems as rational agents that are endowed with the psychological attitudes of belief, desire, and intention" and the problem of using appropriate semantics that give an *intensional* account of these notions. SNePS is an intensional propositional semantic network system [30] that has been used for cognitive modeling, belief representation and reasoning, belief revision, and natural language understanding. A basic principle of SNePS is the Uniqueness Principle—that there be a one-to-one mapping between nodes of the semantic network and concepts (mental objects) about which information may be stored in the network. These concepts are not limited to objects in the real world, but may be various ways of thinking about a single real world object, such as The Morning Star *vs.* The Evening Star *vs.* Venus. They may be abstract objects like properties, propositions, Truth, Beauty, fictional objects, and impossible objects. They may include specific propositions as well as general propositions, and even rules. Any concept represented in the network may be the object of propositions represented in the network giving properties of, or beliefs about it. For example, propositions may be the objects of explicit belief (or disbelief) propositions. Rules are propositions with the additional property that SNIP, the SNePS Inference Package, [17, 29] can use them to drive reasoning to derive additional believed propositions from previous believed propositions.

Plans are also mental objects. We can discuss plans with each other, reason about them, formulate them, follow them, and recognize when others seem to be following them. An AI system, using SNePS as its belief structure, should also be able to do these things. Requiring that the system be able to use a single plan representation for all these tasks puts severe constraints on the design of the representation. For instance, understanding natural language dialogue involving plans requires building plan representations from natural language input. In natural language, the explication of plans generally takes the form of a sequence of rather simple rules (*e.g.*, "If you see John, tell him I'm looking for him," "To pick up a block, you must first clear it"). These rules can contain indefinite and definite

noun phrases and anaphoric references corresponding to typed plan variables. The full plan, including preconditions and effects of its component acts, must be constructed from such a sequence of rules.

Once constructed, a plan must be usable as a specification for the behavior of the agent, and must also be usable by the agent to understand other agents' actions. We are not treating plans as schedules of events for third parties (or multiple agents) [13].

2.3 INTENSIONAL REPRESENTATIONS

We now give an overview of our representations and the motivations that led to them. We use "goal," "plan," "act," and "action" in particular ways, and distinguish among them. A *goal* is a proposition in one of two roles—either the role within another proposition that some *plan* is a plan for achieving that goal (making it true in the then current world), or the role as the object of the *act* of achieving it.

2.3.1 Planning is different from inference

We view a *plan* as a structured individual mental concept, *i.e.*, it is not a proposition or rule that might have a belief status. A plan is a structure of *acts*. (Among which may be the achieving of some goal or goals.) The structuring syntax for plans is a special syntax, differing, in particular, from that used for structuring reasoning rules. This is important both for semantic clarity and to allow a system to be implemented that can both reason and act efficiently. For contrast, consider standard (non-concurrent) Prolog or some arbitrary production rule system. Such a system relies on a semantic ambiguity between the logical & and the procedural **and then**. For example,

(2.3.1) $$p(X) \ :- \ q(X), \ r(X).$$

either means "For any X, $p(X)$ is true if $q(X)$ and $r(X)$ are true" or it means "For any X, to do p on X, first do q on X and then do r on X." Guaranteeing the proper ordering of behavior in the procedural interpretation is only possible by giving up the freedom to reorder, for efficiency, the derivations of $q(X)$ and $r(X)$ in the logical interpretation. The example is made more striking by appending

(2.3.2) $$q(Y) \ :- \ s(Y), \ t(Y).$$

(2.3.3) $$r(Z) \ :- \ s(Z), \ u(Z).$$

and considering the query

(2.3.4) $$?- \ p(a).$$

Under the logical interpretation, it would be efficient for the system to try finding if $s(a)$ holds only once, instead of once when rule 2.3.2 is being used and once when rule 2.3.3 is being used. This is the way SNIP has been implemented (see [17]). However, under the procedural interpretation, it may be perfectly reasonable to perform $s(a)$ twice, so the behavior that optimizes logical reasoning destroys procedural rule following. The fact that SNIP is optimized in this way for reasoning, and so cannot use its reasoning rules as procedural rules, was what originally motivated this project to design a planning/acting component for SNePS.

Believing is a state of knowledge; acting is the process of changing one state into another. Reasoning rules pass a *truth* or a *belief* status from antecedent to consequent, whereas acting

rules pass an *intention* status from earlier acts to later acts. A reasoning rule can be viewed as a rule specifying an act—that of believing some previously non-believed proposition, but the believe action is already included in the semantics of the propositional connective, and, as pointed out above, there is no reason to believe a proposition more than once (unless it's disbelieved in the interim). The distinction between "believing and acting" in SNePS was first outlined in [18].

2.3.2 The distinction between "acts" and "actions"

Lifschitz (1987) attempts to give a semantics of STRIPS by viewing STRIPS as a form of logic and STRIPS operators as rules of inference in this logic. For us, an *act* is a structured individual mental concept of something that can be performed by various actors at various times. This is important for plan recognition—we must be able to recognize that another agent is performing the same act that, if we were performing it, we would be in the midst of carrying out one of a certain number of plans. By the Uniqueness Principle, a single act must be represented by a single SNePS node, even if there are several different structures representing propositions that several different actors performed that act at different times. This argues for a representation of propositions more like that of Almeida [2], rather than like more traditional case-based or frame-based representations. In what we are calling "more traditional representations", there is a structure representing the proposition with slots or arcs to the actor, the action, the object, *etc.* For example, to represent the proposition,

(s1) John walked to the store.

there would be four representational symbols, one for John, one for walking (or PTRANS-ing), one for the store, and one for the proposition itself, and the first three would be connected with the fourth in nearly similar ways at similar distances (measured by path length of arcs or slots). See Figure 2.3.1 for a SNePS representation based on Shapiro & Rapaport (1987). Almeida, however, took seriously that one could follow (s1) by

(s2) Mary did too.

and understand by that that John and Mary performed the same act—that of walking to the store. The representation for (s1) would have to introduce a fifth symbol, for walking to the store, which would be connected to the representation of the proposition at the same distance as the representation of John. Now, however, the symbols for walking and the store would be further from the symbol for the proposition (see Figure 2.3.2). When (s2) is processed, the symbol representing the proposition that Mary walked to the store would be connected to the same symbol for walking to the store used for (s1) (node M7 in Figure 2.3.3). This symbol represents what we are calling an act, and using it in the representation of both propositions follows by the Uniqueness Principle from interpreting (s1) and (s2) as saying that John and Mary performed the same act. Moreover, if the network contains the representation of any plan that involves walking to the (same) store, that same act node

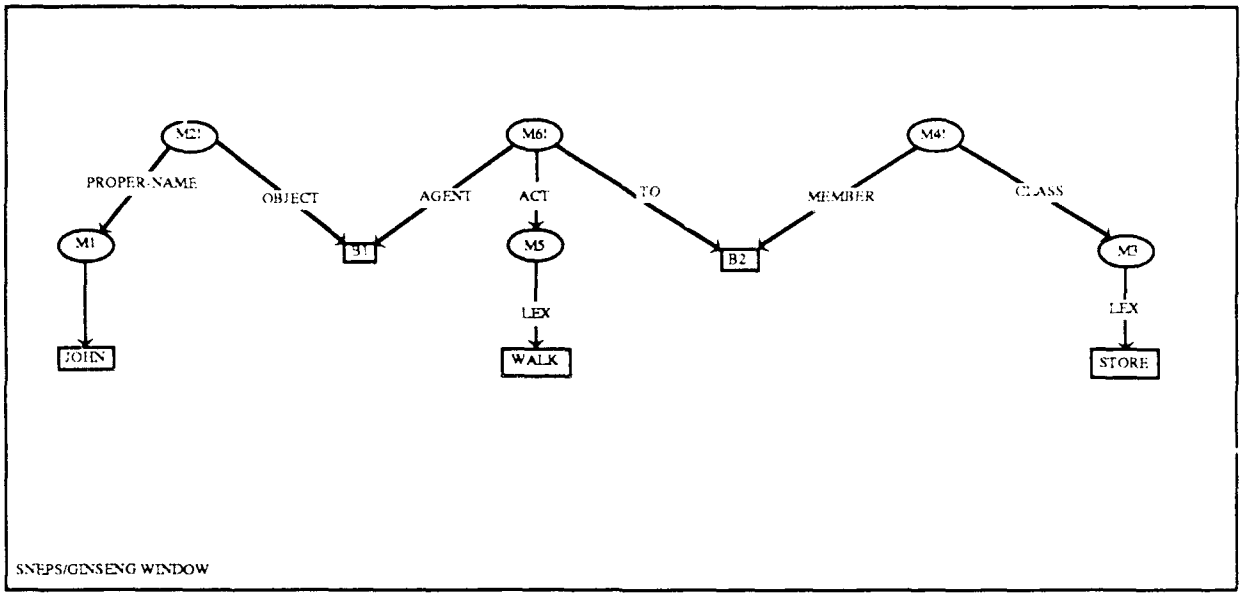

Figure 2.3.1: A traditional representation of "John walked to the store" (ignoring tense). Node B1 represents John; node B2 represents the store; node m6! represents the proposition that John walked to the store.

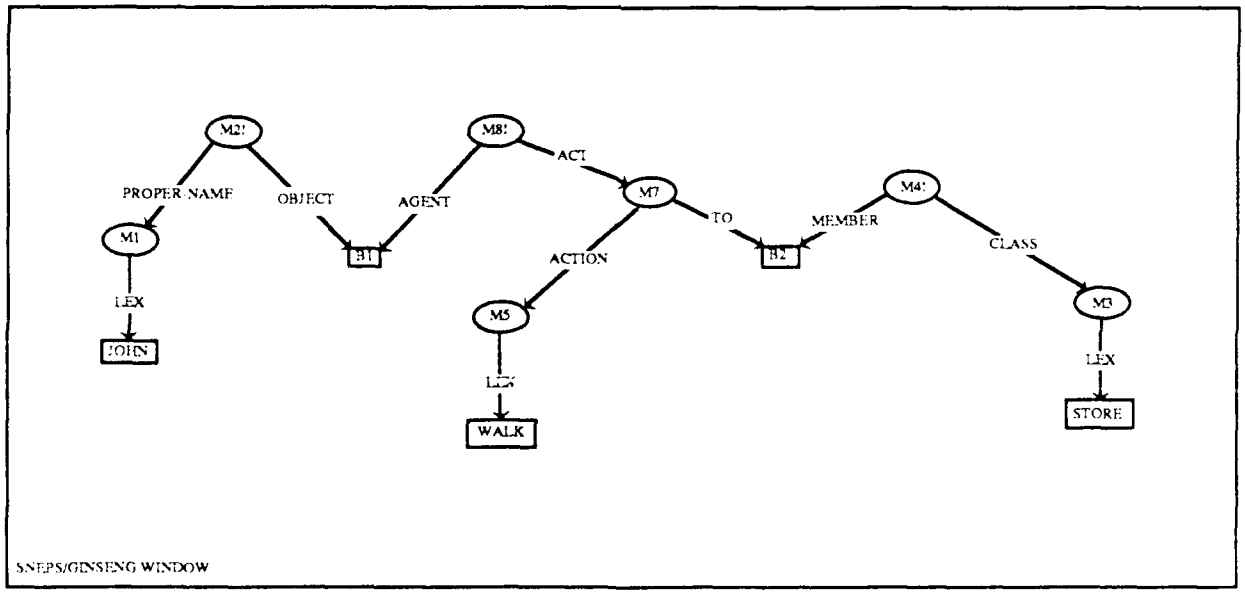

Figure 2.3.2: A representation of "John walked to the store" based on Almeida (1987). Node M7 represents the act of walking to the store; node M8! represents the proposition that John walked to the store.

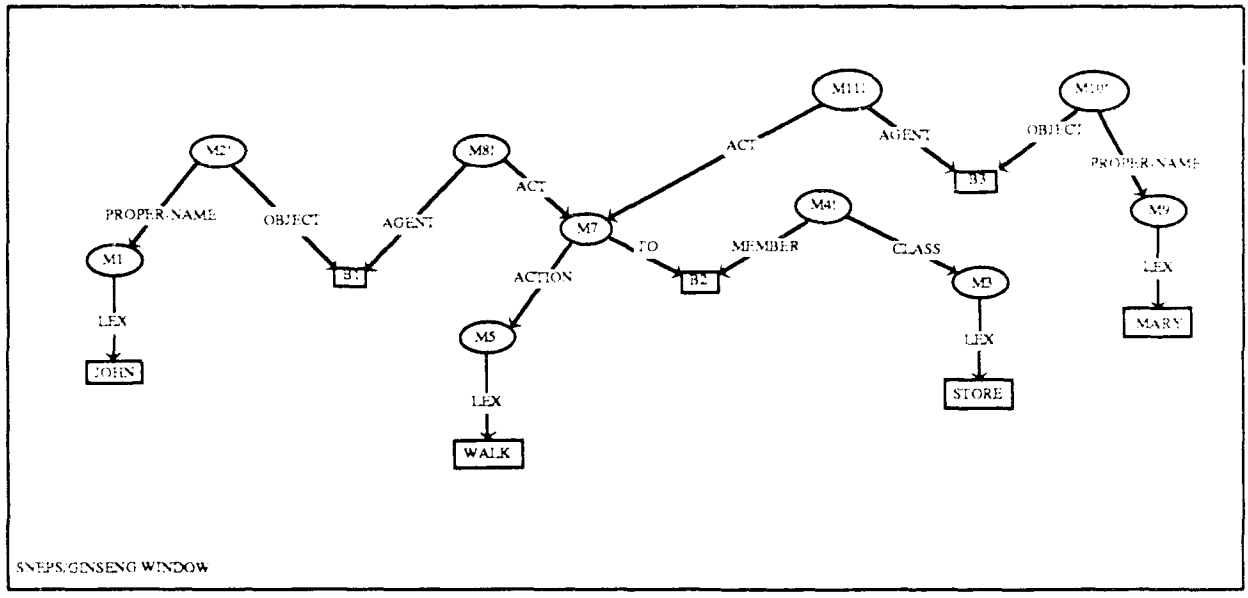

Figure 2.3.3: The network of Figure 2.3.2 with a representation of "Mary did too." added. Node M11! represents the proposition that Mary performed the very same act (represented by node M7) that John did.

would be used in the structure representing that plan (see Figure 2.3.4). Thus, John and Mary are directly connected to a plan that they may be engaged in.

An *action* is that component of an act that is what is done to the object or objects. In (s1) and (s2), the action is walking. Achieving some goal is an act whose action is achieving, and whose object is the particular proposition that is serving as the goal. Unfortunately for our remaining discussion, but consistently with what has gone before, one can only *perform* something that is an act (an action on an appropriate object), so instead of saying "performing an act whose action is x," we will say "performing the action x," and hope the reader will note the distinction between acts and actions.

Our representation of an act is a node with an ACTION arc to a node that represents the action, and OBJECT1, ..., OBJECTN arcs to the required objects of the action. Thus, the general syntax[1] of an act is

Syntax 1: *act* ::= ACTION: *action*
OBJECT1: *object1*
...
OBJECTN: *objectN*

[1]Specific actions might have their objects on differently labeled arcs. For example, in Figures 2.3.2-2.3.4 the WALK action uses a TO arc, and in Figure 2.3.4 the BUY and OBTAIN actions use OBJECT arcs.

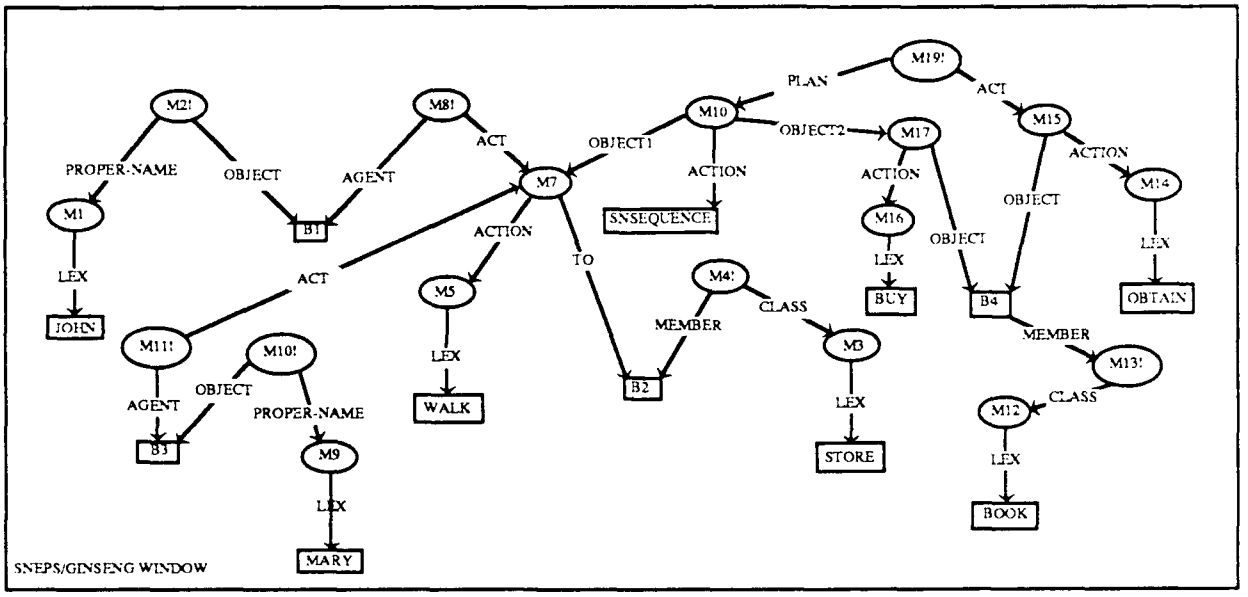

Figure 2.3.4: The network of Figure 2.3.3 with a representation of the proposition that a plan for obtaining some book is to walk to the store and buy the book. Node M19! represents the proposition using a syntax to be introduced in Section 2.5; node M18 represents a sequence of two acts, using a syntax to be introduced in Section 2.4; the first act in that sequence, represented by node M7, is the very same act that John and Mary did.

Semantics 1: *act* is a structured individual node representing the act whose action is *action* and *object1*, ..., *objectN* are the objects of *action*. For example, the SNePSUL (the SNePS User Language) command for building a node representing the act of saying "FOO" is:

```
(build action say object1 FOO)
```

2.3.3 Primitive and Complex actions

Any behaving entity has a repertoire of *primitive actions* it is capable of performing. We will say that an act whose action is primitive is a *primitive act*. That an action is primitive is a belief held by SNACTor after we tell it. The belief is represented in the form of an assertion saying that the action is a member of the class of primitive actions. This is similar to the MEMBER-CLASS proposition used by CASSIE [30]. For example, the SNePSUL command for asserting that saying is a primitive action is

```
(assert member say class primitive)
```

Non-primitive acts, which we will term *complex*, can only be performed by decomposing them into a structure of primitive acts, the syntax of which is the same procedural syntax as used in plans. That some plan p is a plan for carrying out some complex act a, is a proposition we can assert to SNACTor using the following representation:

Syn. 2: *plan-act-proposition* ::= ACT: a
PLAN: p

Sem. 2: *plan-act-proposition* is a proposition node that represents that p is a plan for carrying out act a.

p is a structure of acts. The structuring syntax for plans is described in terms of control actions which are described later. That some plan p is a plan for achieving some goal g is also a proposition we can assert to SNACTor:

Syn. 3: *plan-goal-proposition* ::= GOAL: g
PLAN: p

Sem. 3: *plan-goal-proposition* is a proposition node that represents that p is a plan for achieving goal g.

g is expressed as a domain specific proposition. Examples of these are given in a later section (see 2.5).

When the time comes for the agent to perform a complex act, it must find a plan that decomposes it. Using the above representations SNACTor may be told such plans. SNACTor is also capable of doing classical planning in case it does not already know any decompositions for a complex act. This is discussed later.

2.3.4 Pre– and post–conditions

The remaining notions we must consider are preconditions and effects (postconditions). Whether we think of them as pre- and post-conditions of plans or of acts is irrelevant since plans are kinds of acts. A pre-(post-)condition is just a proposition that must be (will be) true or false before (after) an act is performed. But the proposition that a proposition p is false is itself a proposition, so we can say that a pre-(post-)condition is a proposition that must be (will be) *true* before (after) an act is performed. (We will rely on SNeBR, the SNePS Belief Revision System [16] to remove inconsistent beliefs after believing the effects of an act.) We have thus reduced the storage of pre- and post-conditions to two simple kinds of propositions:

Syn. 4: *precondition-proposition* ::= ACT: a
PRECONDITION: p

Sem. 4: *precondition-proposition* is a node that represents that the pre-condition of some act a is the proposition p.

For example,

```
(assert forall $block
        ant (build member *block class block)
        cq (build act (build action pickup object1 *block)
                  precondition (build property clear object *block)))
```

is the SNePSUL command to assert that before picking up any block first make sure that it is clear (*i.e.* there is nothing on top of it).

Syn. 5: *postcondition-proposition* ::= ACT: a
EFFECT: p

Sem. 5: *postcondition-proposition* is a node that represents that the post-condition of some act a is the proposition p.

For example,

```
(assert forall $block
        ant (build member *block class block)
        cq (build act (build action pickup object1 *block)
                  effect (build property holding object *block)))
```

is the SNePSUL command to assert that after picking up any block, it is being held.

Thus, effects and preconditions of an act are represented in the same way as other beliefs about other mental objects; we do not need a special data structure (or an operator formalism) for acts in which pre- and post-conditions are special fields. Such a representation also enables us to assert context-dependent effects of actions [38]; *i.e.*, the effects of doing some action are determined by the context in which the action is performed. For example,

```
(assert forall ($block $support)
        &ant ((build member *block class block)
              (build member *support class support)
              (build rel on arg1 *block arg2 *support))
        cq (build act (build action pickup object1 *block)
                  effect (build property clear object *support)))
```

asserts that if a block is on some support then after you pick up the block the support is clear. The scope of the context being referred to is the set of beliefs held by the system at the time the action is about to be performed. Using rules like these, context-dependency is guaranteed by ensuring that the effect is conditional on the antecedents being true before the act is performed. This is a more natural way of modeling actions and avoids the need for specifying multiple operators for doing the same action in different situations, which is a major criticism of earlier planners [5]. In Section 2.6.4 we demonstrate this feature.

2.3.5 Types of actions

We discussed three kinds of acts: a *primitive* act is unstructured and is in the repertoire of the agent; a *complex* act is unstructured—to perform it, the agent must find a plan for it; a *plan* is a structured act—the structure determines how the agent performs the component acts.

The structure of a plan can determine how the agent performs the component acts, because the structure, itself, is a primitive action.

Primitive actions fall into three classes:

- external actions that affect the world;
- mental actions that affect the agent's beliefs;
- control actions that affect the agent's intentions.

External actions are domain specific actions like pickup, putdown etc. in the Blocksworld. The two mental actions that we have are *believing* a proposition, and *disbelieving* a proposition. Our repertoire of control actions includes *sequencing, conditional, iterative, and achieve* actions. A *sequencing* action represents the agent's *intention* to perform its object actions in a given sequence. The *conditional* and *iterative* actions are modeled after Dijkstra's guarded-if and guarded-loop commands respectively [3]. The *achieve* action deduces plans for achieving some proposition and forms the intention of performing one of them. The *conditional* and *iterative* control actions enable the specification of non-linear partial plans. We can also have an explicit representation of non-linear plans using an appropriate control action. We have also designed a control action that can be used for posting constraints on plan variables (as in [38]).

2.3.6 Modeling external effects of actions

As mentioned above, external actions are domain specific actions that affect the outside world. For example, if the agent has an arm and is asked to pick up a block, the arm actually moves to the block, grasps it, and then lifts it up. Depending on the set of interfaces provided to the agent (like an arm, a speech synthesizer, etc.) we need to be able to carry out the action in the external world. This is done by writing Common Lisp functions that access the external interface. For instance, we can model the external effects of the 'say' action by driving a speech synthesizer or by simply printing the message on the screen. The `define-primaction` function enables us to do this. Thus, to model saying something by printing it on the screen, we will have

```
(define-primaction say (n)
  "n is the node representing the act of saying. The node at the
   end of object1 arc is printed. choose.ns and pathfrom are
   SNePS interface functions to access parts of a structured node."
   (format t " ~A " (choose.ns (pathfrom 'object1 n))))
```

Thus when the agent executes the action represented by

```
(build action say object1 FOO)
```

the above code for `say` is executed, resulting in 'FOO' appearing on the screen. How an action gets scheduled to be executed is discussed in the next section. Section 2.9.1 has some more examples of modeling primitive blocksworld actions.

2.4 THE PLANNING PARADIGM

Besides having a current set of beliefs about the world, the system also has beliefs about plans for achieving goals, and about how complex actions can be decomposed into partial plans. The overall architecture of the system is similar to that of the PRS system [7] The acting executive (called an *interpreter* or a *reasoning mechanism* in PRS) manipulates these components. It maintains an acting queue (referred to as a *process stack* in PRS) that contains all the scheduled actions to be performed as a part of some plan and so it represents the system's *intentions.* The system can also form its own intentions in response to changing beliefs. SNIP, the SNePS inference package is used for several tasks: to find plans for complex tasks; as part of the achieve action, to find a plan to achieve some goal; and also as the *truth criterion* (also called the *question answering procedure*, see [4]). Hence, it is used as the *plan decision procedure* in our system. SNIP is implemented on a simulated multi-processing system. In the future, we will be able to do hypothetical reasoning using SNeBR [15] for state-based plan projection. SNeBR is currently being used as a Truth Maintenance System. This facilitates the implementation of *mentalactions* and *conditionalplans* as discussed below.

2.4.1 The acting executive

We want the system to carry out plans, as well as to discuss them, reason about them, and recognize them. Certainly, since the system is currently without eyes, hands, or mobility, its repertoire of primitive actions is small, but, for now, as shown above, we can simulate other actions by appropriate printed messages. SNACTor, the acting system is composed of a queue of acts to be carried out, and an acting executive. The queue of acts represents the system's intentions for carrying out the acts on the queue in that order. Intentions are formed by either an explicit request from a user to do something, or by committing to a plan that needs to be executed to fulfill a complex act or a goal. Explicit requests are made using the `perform` command. For example,

```
(perform (build action say FOO))
```

is an explicit request to the system to say 'FOO'. The act is put on the `act-queue` and the acting executive takes charge. Currently, the acting executive is the following loop:

```
while act-queue is not empty do
      if   the first-act on the act-queue has preconditions
           and they are not currently satisfied
           then insert the achieving of them on the front of the act-queue
      else remove the first-act from the act-queue;
           deduce   effects of the first-act,
                    and insert the believing of them on the front
                    of the act-queue;
           if   the first-act is primitive
                then perform it
           else deduce plans for carrying out the first-act
                (using SNIP and available rules),
                and insert the act of doing one of them
                on the front of the act-queue
           end if
      end if
end while
```

Notice that the effects of the act about to be performed are retrieved and scheduled to be believed before the act is actually performed. This guarantees that proper effects of the act are retrieved depending on the context that exists at that time. This flexibility in dynamically determining the effects of acts is what enables us to avoid having multiple operators for the same action.

When preconditions for an act exist and some of them are found not to be true, we schedule the achieving of all of them on the queue. The intention to perform the act is now pushed behind the intention to achieve these preconditions. Once all the preconditions are achieved, and we are ready to perform the act, they are checked again (just in case achieving some precondition renders another one false).

From the above loop, it can be seen that we are assuming that a plan will be found for every complex act, and that every act will be successful.

2.5 SYNTAX AND SEMANTICS OF CONTROL ACTIONS

We are now ready to examine the syntax and operational semantics of our current set of control actions.

Syntax 6: *sequence* ::= ACTION: SNSEQUENCE
OBJECT1: *act1*
OBJECT2: *act2*

This means that a *sequence* act is represented by a node with an ACTION arc to the node SNSEQUENCE, an OBJECT1 arc to an *act* node, and an OBJECT2 arc to another *act* node.

Semantics 6: *act2* is inserted on the front of the act queue, and then *act1* is inserted in front of it.

For example, a plan to get a block on a support is to pick it up and then put it down on the support. This can be derived using the *plan-goal-proposition* and *snsequence* as

```
(assert forall ($block $support)
        &ant ((build member *block class block)
              (build member *support class support))
        cq (build plan (build action snsequence
                              object1 (build action pickup object1 *block)
                              object2 (build action putdown
                                       object1 *block object2 *support))
                   goal (build rel on arg1 *block arg2 *support)))
```

Another example of a sequence is represented by node M18 in Figure 4. Since either or both of *act1* and *act2* can themselves be *snsequence* acts, we have a general structure for plans of sequential actions.

Syntax 7: *do-one* ::= ACTION: DO-ONE
OBJECT1:{*acti*}

This means that a *do-one* act is represented by a node with an ACTION arc to the node DO-ONE, and OBJECT1 arcs to an arbitrary number of *act* nodes.

Semantics 7: Chooses one *acti* and puts it on the front of the act queue. As currently implemented, the choice is arbitrary.

For example, an act of giving an arbitrary greeting by saying "HELLO" or "JAMBO" or "G-DAY" can be expressed as

```
(build action do-one
       object1 ((build action say object1 HELLO)
                (build action say object1 JAMBO)
                (build action say object1 G-DAY)))
```

Syntax 8: *do-all* ::= ACTION: DO-ALL
OBJECT1: {*acti*}

Semantics 8: Forms the intention of doing all the *acti* by placing them on the front of the act queue in some unspecified order.

For example, an agent's "things-to-do-today" list can be represented using such an act as

```
(build action do-all
       object1 ((build action buy object BOOK)
                (build action pay object PHONE-BILL)
                (build action see object NIAGARA-FALLS)))
```

Syntax 9: *conditional* ::= ACTION: SNIF
OBJECT1: {CONDITION: *propositioni*
THEN: *acti*}

This means that a *conditional* act is represented by a node with an ACTION arc to the node SNIF, and OBJECT1 arcs to an arbitrary number of nodes, each with a CONDITION arc to a *proposition* node and a THEN arc to an *act* node.

Semantics 9: If no *proposition* is true, does nothing. Otherwise, a *do-one* act whose objects are all the *acti* having their corresponding *propositioni* true is put on the front of the act queue. (Based on Dijkstra's guarded if [3].)

For example, an act of saying "HELLO" contingent upon having permission can be expressed as

```
(build action snif
       object1 (build condition (build have permission)
                       then (build action say object1 HELLO)))
```

Syntax 10: *iteration* ::= ACTION: SNITERATE
OBJECT1: {CONDITION: *propositioni*
THEN: *acti*}

Semantics 10: If no *proposition* is true, does nothing. Otherwise, puts on the front of the act queue a *sequence* whose OBJECT1 is a *do-one* act whose objects are all the *acti* having their corresponding *propositioni* true, and OBJECT2 is the *iteration* node itself. (Based on Dijkstra's guarded loop [3].)

For example, the act of repeatedly saying "HELLO" contingent upon having "hello-permission" and saying "THERE" contingent upon having "there-permission" can be expressed as

```
(build action sniterate
       object1 ((build condition (build have hello-permission)
                        then (build action snsequence
                          object1 (build action say object1 HELLO)
                          object2 (build action forget
                                 object1 (build have hello-permission))))
                (build condition (build have there-permission)
                        then (build action snsequence
                          object1 (build action say object1 THERE)
                          object2 (build action forget
                                 object1 (build have there-permission))))))
```

Syntax 11: *achieve* ::= ACTION: ACHIEVE
OBJECT1: *proposition*

Semantics 11: If *proposition* is true, does nothing. Otherwise, deduces plans for achieving *proposition*, chooses one of them, and puts it on the front of the act queue.

For example, in order to achieve a state in which BLOCKA is clear we'll have the act

```
(perform (build action achieve
                object1 (build property clear object BLOCKA)))
```

Thus, we can write plans for achieving goals as well as plans for decomposing a complex act. The domain normally determines the kinds of plans required (*i.e.*, goal-based or act-decomposition based or both). However, as we will see, in the case of the blocksworld, and possibly in other domains, it may become hard to distinguish between something that characterizes a state and something that expresses an act. For example, "Clear BLOCKA" could be interpreted as a command to perform the act of clearing BLOCKA or a goal to achieve a state in which BLOCKA is clear. In any case, if required, we can model and use both interpretations.

2.6 MENTAL ACTIONS

As mentioned earlier we are using SNeBR as a Truth Maintenance system underlying the knowledge-base. Beliefs currently held by the system are maintained in a belief context. The ATMS ensures consistency of beliefs at all times. Effects of acts lead to adding or deleting of new beliefs. We use the SNeBR operations *add – to – context* and *remove – from – context* to add or remove new beliefs as a consequence of performing some act. The calls to these operations are used to model the two mental actions of *believing* and *disbelieving*.

add – to – context adds a belief to the hypothesis set of the context. Similarly *remove – from – context* removes a belief from the hypothesis set of the context. SNeBR ensures that all derived beliefs that used the removed hypothesis as an assumption in their derivation are also automatically removed. This is especially useful in the implementation of conditional plans.

2.7 CONDITIONAL PLANS

A conditional plan, such as

> "If a block is on a support then a plan to achieve that the support is clear is to pick up the block and then put the block on the table."

is represented in SNePS as a rule approximately like the Predicate Calculus rule

$$\forall x, y[Block(x) \wedge Support(y) \wedge On(x, y) \Rightarrow GoalPlan(Clear(y), Sequence(Pickup(x), Put(x, Table)))].$$

In a situation in which block A is on block B, and the system must clear B, it will derive and store the plan,

$$GoalPlan(Clear(B), Sequence(Pickup(A), Put(A, Table))),$$

which says that a plan to clear B is to pick up A and put it on the table.

Since this plan is stored, it would seem that it would be retrieved as a plan for clearing B in some later situation when C, for example, is on B, and this would be wrong. However, our system is implemented in SNePS-2.1, which includes the assumption-based SNePS Belief Revision System [16]. The plan

$$GoalPlan(Clear(B), Sequence(Pickup(A), Put(A, Table))),$$

is derived based on the assumptions $Block(A)$, $Support(B)$, and $On(A, B)$. As soon as A is picked up, the assumption $On(A, B)$ is removed from the current context, and the plan is unavailable until A is put back on top of B. Thus, the representation of conditional plans is correct in systems that include an appropriate belief revision mechanism.

2.8 NATURAL LANGUAGE PLAN INSTRUCTION

During the work described in this report, it became clear that natural language sentences about planning could be classified into groups, with associated syntactic natural language markers. In the current domain, Blocksworld, we classify the sentences as follows:

1. **Domain description** These types of sentences are simple declarative statements about the state of the domain, *e.g.*, "A is a block", "A is on the table".

2. **Constraint description** These types of sentences are statements about the constraints on objects in the domain, (*e.g.*, "If a block is on a support then the block is not on another support") independent of a the agents capabilities.

3. **Act/Plan description** These types of sentences define the types of actions the agent (SNACTor) is capable of performing, their associated preconditions and effects as well as how they are composed to build complex plans.

4. **Performative requests** These types of sentences are simple imperative requests to perform an action, *e.g.*, "Pick up A".

Type (1) sentences generaly are simple copular clauses, or 'There-is' introductory sentences (*e.g.*, "There is a table"). Noun phrases are generally definite, with proper name reference being the rule, rather than the exception. Type (4) sentences are simple imperative requests whose object noun phrases may be definite or indefinite, *e.g.*, ("Pile A on B on C", "Put A on a block").

Sentences of type (2) and (3) are rule-like (similar to FOPL rules), in their structure and use of variables. In our domain, constraint description sentences map directly to FOPL rules; *e.g.*, "If a block is on a support then the block is not on another support" becomes

$$\forall x, y, z[(Block(x) \wedge Block(y) \wedge Block(z) \wedge On(x,y)) \Rightarrow \neg on(x,z)]^1.$$

Act/Plan description sentences have less direct mapping, but in both the treatment of definite/indefinite noun phrases is the same. In the context of a sentence of these types an indefinite noun phrase introduces a new variable of the type associated with the common noun of the noun phrase. Definite noun phrases refer back to explicitly introduced (via an indefinite noun phrase) variables in the same sentence. The syntax of these types of sentences is well marked (by words like "before", "after", "then") and is highly decomposable into simpler clauses of types (1) and (4). For instance a plan to pile three blocks consists of three repeated type (4) sentences (*e.g.*, "put the first block on the table", "put the second block on the first block", etc.) expressed in the appropriate order with explicit sequence separators ("... and then ..."). Similarly, preconditions and effects have antecedent and

[1]The conjunct $x \neq y$ is not needed in the antecedent of this rule because SNIP uses the Unique Variable Binding Rule (UVBR) [26] which prevents x and y from binding to the same term.

consequent clauses of type (1). In both type (2) and (3) sentences, the noun phrase positions associated with component clauses are generally variables, although definite reference to objects in the domain ("the table") also occur.

In this domain, sentences about plans display a useful compositionality, in that more complex types of sentences (describing more complex types of knowledge) can be built from the simple clauses associated with describing a domain and performing acts, where they differed was in the treatment of definite/indefinite noun phrases as referring to and introducing variables, respectively.

2.9 USING PLANS

In this section we will demonstrate how to model actions and do planning in the domain of a blocksworld. We will model a blocksworld where there is a robot arm that can do two primitive actions (*pickup*, and *put*). In an earlier report [31] we used a model based on [19] that had four primitive actions (*pickup*, *putdown*, *stack*, and *unstack*). [31] contains a SNePSUL version of a blocksworld with two primitives. In this report we will demonstrate the capabilities of our system through natural-language interactions. We will also show how the model can easily be extended through natural language interactions.

2.9.1 The natural language front-end to a blocksworld

The natural language understanding component is implemented in a Generalized ATN grammar and is used for analyzing sentences and for generating English responses. SNACTor begins with an empty knowledge-base. In the role of informant, we interact with SNACTor using English sentences about the domain, instructing SNACTor about the various actions that it can do, and how to solve problems in that domain. The input sentences are analyzed using a domain-specific grammar, the results of which are new beliefs in the knowledge-base. A natural language generation grammar takes the new beliefs and expresses them back in English to show SNACTor's understanding to the informant. Requests to do some action are sent to an acting executive that may then generate and execute a plan to fulfill the request. The informant may also ask questions about plans and the way the system would solve various problems.

As mentioned above, our model of the blocksworld consists of some rectangular blocks on a table. The blocks are identified by names (like A, B, C, *etc.*). Beliefs about the current state of the blocksworld are held using the following predicates:

clear(x) **x** is clear (*i.e.*, there is nothing on top of it).

on(x,y) **x** is on top of **y**.

held(x) **x** is currently being held.

These are represented in SNePS using the *property-object* proposition case-frames and the *rel-arg1-arg2* proposition case frames.

Syntax. 10: *property-object-prop* ::= PROPERTY: *property*
OBJECT: *object*

Semantics. 10: *property-object-prop* is a proposition node representing the proposition that *object* has property *property*.

This can be used to represent the **held** and **clear** predicates. For example, to assert that block A is clear and block B is being held,[1] we have

```
(assert property clear object A)
(assert property held object B)
```

Syntax. 11: *rel-arg1-arg2-prop* ::= REL: *relation*
ARG1: *object1*
ARG2: *object2*

Semantics. 11: *rel-arg1-arg2-prop* is a proposition node representing that *relation* holds between *arg1* and *arg2*.

This can be used to represent the **on** predicate. For example, to assert that block C is on top of block A, we have

```
(assert rel on arg1 C arg2 A)
```

Thus when we tell the system in English

```
C is a block. C is clear. C is on the table.
A is a block. A is clear. A is on C.
```

the sentences are analyzed by the natural language understanding system and the following SNePSUL commands are executed to assert the beliefs expressed by the sentences

```
(ASSERT MEMBER (BUILD LEX C)
        CLASS (BUILD LEX block))
(ASSERT PROPERTY (BUILD LEX clear)
        OBJECT (BUILD LEX C))
(ASSERT REL (BUILD LEX on)
        ARG1 (BUILD LEX C)
        ARG2 (BUILD LEX table))
(ASSERT MEMBER (BUILD LEX A)
        CLASS (BUILD LEX block))
(ASSERT PROPERTY (BUILD LEX clear)
        OBJECT (BUILD LEX A))
(ASSERT REL (BUILD LEX on)
        ARG1 (BUILD LEX A)
        ARG2 (BUILD LEX C))
```

[1] the fact that A and B are blocks is asserted in the knowledge-base using the *member – class* proposition [30]

The LEX arcs are used by the system to represent that the name of the property (or rel) is expressed in English (*i.e.*, lexically) as "clear" or "on" etc. Thus appropriate morphological analyses and syntheses can be applied to them during understanding and generation. This is discussed in more detail in [30].

The generation component then takes over and expresses the resulting propositions in English, which forms the system's response to demonstrate its understanding of what was said

```
I understand that C is a block.
I understand that C is clear.
I understand that C is on table.
I understand that A is a block.
I understand that A is clear.
I understand that A is on C.
```

Modeling primitive actions

We can ask the agent to perform the following primitive actions

pickup(x) This is an action specifying that the agent is to pick up **x** where **x** is some block.

put(x,y) This action is a request to put **x** on **y**. **x** is some specified block and **y** is some specified support (blocks and tables are supports).

The external interface to the blocksworld is simulated in the form of an appropriate message. The system also creates a graphics window that graphically shows the state of the blocksworld at any given instant. The primitive actions are appropriately interfaced so as to simulate their external effects in the graphics window. The following Common Lisp code specifies external effects of these actions:

```
(define-primaction pickup (n)
  (format t "~&Now  doing: Pickup ~A.~%"
          (choose.ns (pathfrom '(object1 lex) (node n))))
  (bw-pickup
    (eval (node-na (choose.ns (pathfrom '(object1 lex) (node n)))))))

(define-primaction put (n)
  (format t "~&Now doing: Put ~A on ~A.~%"
          ...)
  (if (eval `(sneps:isbase.n ,(choose.ns (pathfrom 'object2 n))))
      (bw-putdown (eval (node-na block1)) table)
      (bw-putdown (eval (node-na block1)) (eval (node-na block2)))))
```

2.9.2 An annotated example

We will now show how the system is instructed about a blocksworld. We begin by getting into SNePS

```
> (sneps)

 Welcome to SNePS-2.1
 Copyright 1984, 1988, 1989 by
 Research Foundation of State University of New York
 8/8/1989 23:52:55

*
```

At this point we will load SNACTor, the parser, the ATN grammar, the lexicon, and define the required arcs and paths. Now, we are ready to get into the parser and start telling the system about our blocksworld. We begin by informing it about the primitive actions.[2]

$*(\wedge(parse - 1))$

```
ATN parser initialization...
Input sentences in normal English orthographic convention.
May go beyond a line by having a space followed by a <CR>
To exit parser, write ^end.
```

There is a table. The table is a support. Blocks are supports.

```
I understand that B1 is a table.
I understand that a table is a support.
I understand that blocks are supports.
```

Picking up is a primitive action. Before picking up a block the block must be clear. After picking up a block the block is not clear and the block is held. If a block is on a support then after picking up the block the block is not on the support and the support is clear.

```
I understand that pickup is a primitive.
I understand that before performing pickup on a block,
the block must be clear.
I understand that after performing pickup on a block,
the block is not clear.
I understand that after performing pickup on a block,
the block is held.
```

[2]Sentences in italics are inputs by the user. The rest are all system responses

```
I understand that if a block is on a support then
after performing pickup on the block, the block is not on the support.
I understand that if a block is on a support then
after performing pickup on the block, the support is clear.
```

Putting is a primitive action. Before putting a block on a support the block must be held and the support must be clear. After putting a block on a support the block is not held and the block is clear and the block is on the support. After putting a block on another block the latter block is not clear.

```
I understand that put is a primitive.
I understand that before performing put on a block and a support,
the block must be held.
I understand that before performing put on a block and a support,
the support must be clear.
I understand that after performing put on a block and a support,
the block is not held.
I understand that after performing put on a block and a support,
the block is clear.
I understand that after performing put on a block and a support,
the block is on the support.
I understand that after performing put on a block and another
block, the latter block is not clear.
```

At this point we have successfully told the system about the two primitive actions and their preconditions and effects. We now tell it about some plans.

A plan to achieve that a block is held is to pick up the block.

```
I understand that a plan to achieve that a block is held is
by performing pickup on the block.
```

A plan to achieve that a block is on a support is to put the block on the support.

```
I understand that a plan to achieve that a block is on a support is
by performing put on the block and the support.
```

If a block is on a support then a plan to achieve that the support is clear is to pick up the block and then put the block on the table.

```
I understand that if a block is on a support
then a plan to achieve that the support is clear is
by performing pickup on the block
and then performing put on the block and a table.
```

A plan to pile a block on another block on a third block is to put the third block on the

table and then put the second block on the third block and then put the first block on the second block.

```
I understand that a plan for performing pile on a block
and another block and another block is
by performing put on the third block and a table
and then performing put on the second block and the third block
and then performing put on the first block and the second block.
```

Now the system knows plans for decomposing some complex acts (like piling) as well as plans for achieving some goals. We are now ready to describe a state of blocks in the blocks world to play with.

A is a block. B is a block. C is a block. C is clear. C is on the table. B is clear. B is on the table. A is clear. A is on the table.

```
I understand that A is a block.  I understand that B is a block.
I understand that C is a block.  I understand that C is clear.
I understand that C is on a table.  I understand that B is clear.
I understand that B is on a table.  I understand that A is clear.
I understand that A is on a table.
```

We now ask it to do some simple things.

Pick up C.

```
I understand that you want me to perform pickup on C.
Now doing:  Pickup C.
Disbelieve C is clear.
Believe C is held.
Believe a table is clear.
Disbelieve C is on a table.
```

Put C on A.

```
I understand that you want me to perform put on C and A.
Now doing:  Put C on A.
Believe C is clear.
Disbelieve C is held.
Believe C is on A.
Disbelieve A is clear.
```

At this point the world looks as shown in Figure 2.9.1 We now ask it to pile A on B on C. SNACTor also has a built-in trace facility that allows it to speak what it is currently doing (*i.e.* decomposing complex acts, searching for preconditions, etc.). It shows the system's current intentions and what it is about to do.

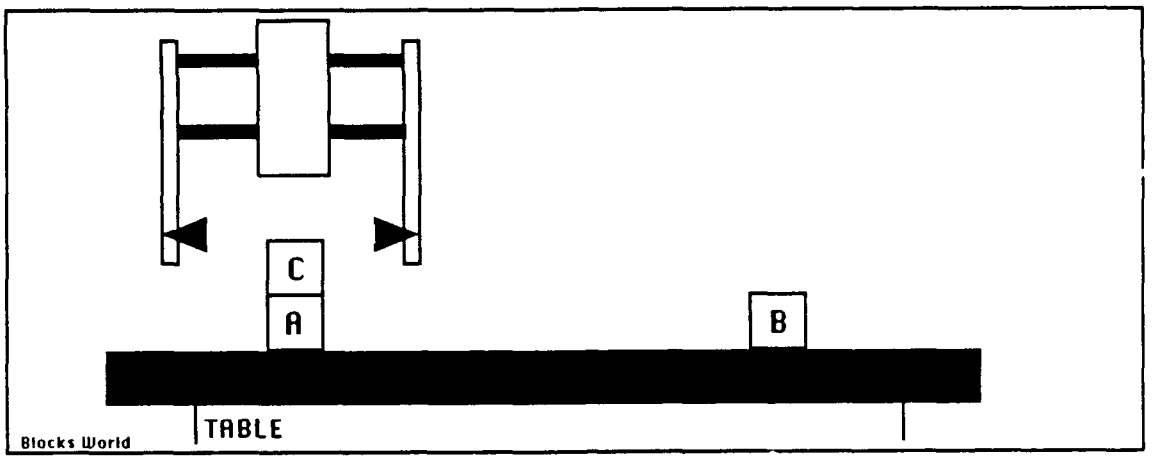

Figure 2.9.1: A state of the blocksworld.

Pile A on B on C.

```
I understand that you want me to perform pile on A and B and C.
```

SNACTor now goes into its acting executive which realizes that piling is a complex act and it needs to be decomposed. This is where it uses beliefs acquired from our earlier dialog and finds decompositions:

```
I intend to do
performing pile on A and B and C.

About to do
performing pile on A and B and C.

I deduced
a plan for performing pile on A and B and C is
  by performing put on C and a table
  and then performing put on B and C
  and then performing put on A and B.
```

This forms the system's current set of intentions and it now sets out to fulfill them.

```
I intend to do
performing put on C and a table.
performing put on B and C.
performing put on A and B.

About to do
performing put on C and a table.
```

Before it performs any action it checks to see if there are any preconditions and if they are satisfied.

```
The act performing put on C and a table.
has the following preconditions:
C is held.
a table is clear.
They are not satisfied.

Want to achieve C is held.

I deduced
a plan to achieve that C is held is by performing pickup on C.

I intend to do
performing pickup on C.
performing achieve on a table is clear.
performing put on C and a table.
performing put on B and C.
performing put on A and B.

About to do
performing pickup on C.

The act performing pickup on C.
has a precondition:C is clear.
It is satisfied.

Now  doing: Pickup C.
Disbelieve C is on A.
Believe A is clear.
Disbelieve C is clear.
Believe C is held.
Want to achieve a table is clear.
Already Achieved.

I intend to do
performing put on C and a table.
performing put on B and C.
performing put on A and B.

About to do
performing put on C and a table.

The act performing put on C and a table.
```

```
has the following preconditions:
C is held.
a table is clear.
They are satisfied.
Now doing: Put C on table.
Believe C is clear.
Believe C is on a table.
Disbelieve C is held.

I intend to do
performing put on B and C.
performing put on A and B.

About to do
performing put on B and C.

The act performing put on B and C.
has the following preconditions:
C is clear.
B is held.
They are not satisfied.

Want to achieve C is clear.
Already Achieved.

Want to achieve B is held.

I deduced
a plan to achieve that B is held is by performing pickup on B.

I intend to do
performing pickup on B.
performing put on B and C.
performing put on A and B.

About to do
performing pickup on B.

The act performing pickup on B.
has a precondition:B is clear.
It is satisfied.
```

```
Now  doing: Pickup B.
I already believe a table is clear.
Disbelieve B is clear.
Believe B is held.
Disbelieve B is on a table

I intend to do
performing put on B and C.
performing put on A and B.

About to do
performing put on B and C.

The act performing put on B and C.
has the following preconditions:
C is clear.
B is held.
They are satisfied.

Now doing: Put B on C.
Disbelieve B is held.
Believe B is on C.
Believe B is clear.
Disbelieve C is clear.

I intend to do
performing put on A and B.

About to do
performing put on A and B.

The act performing put on A and B.
has the following preconditions:
A is held.
B is clear.
They are not satisfied.

Want to achieve A is held.

I deduced
a plan to achieve that A is held is by performing pickup on A.
```

```
I intend to do
performing pickup on A.
performing achieve on B is clear.
performing put on A and B.

About to do
performing pickup on A.

The act performing pickup on A.
has a precondition:A is clear.
It is satisfied.

Now  doing: Pickup A.
Believe A is held.
Disbelieve A is on a table.
I already believe a table is clear.
Disbelieve A is clear.

Want to achieve B is clear.
Already Achieved.

I intend to do
performing put on A and B.

About to do
perfc-ming put on A and B.


The act performing put on A and B.
has the following preconditions:
A is held.
B is clear.
They are satisfied.
Now doing: Put A on B.
Disbelieve A is held.
Believe A is on B.
Believe A is clear.
Disbelieve B is clear.
```

2.9.3 Discussing plans

We have seen how we can instruct SNACTor about planning in a domain and how we can describe situations to it and subsequently ask it to do things by using the plans it derives. We can also ask questions about plans and the various beliefs that it holds. For example,

Is A on B?

```
Yes, A is on B.
```

Is A on C?

```
I really don't know if A is on C.
```

The answer to the second query above is inconclusive because there is no way for the system to confirm or deny the queried fact. To be able to do so, we can instruct the system about domain-specific rules in English. For example, to answer the above query conclusively it needs the following rule:

If a block is on a support then the block is not on another support.

```
I understand that if a block is on a support
then the block is not on another support.
```

Is A on C?

```
No, A is not on C.
```

Similarly we can ask the system to answer questions involving plans. SNIP, the plan decision procedure, is used to derive an appropriate plan and respond to the query. For example,

How would you pile A on C on B?

```
I understand that a plan for performing pile on A and C and B is
by performing put on B and a table
and then performing put on C and B
and then performing put on A and C.
```

Notice that in this case a plan is derived but not executed. So far we have demonstrated that the system is able to interact with the user about its beliefs, it can understand natural language domain descriptions of domain-specific rules and plans and acts, it can use the domain rules to answer queries about the domain, it can also use the acquired beliefs to plan and act in that domain.

2.9.4 Extending the blocksworld

Assume that the system has just finished making a pile as above and a new block D has been introduced and the system now has beliefs about it too (see Figure 2.9.2). Let us now ask the system to

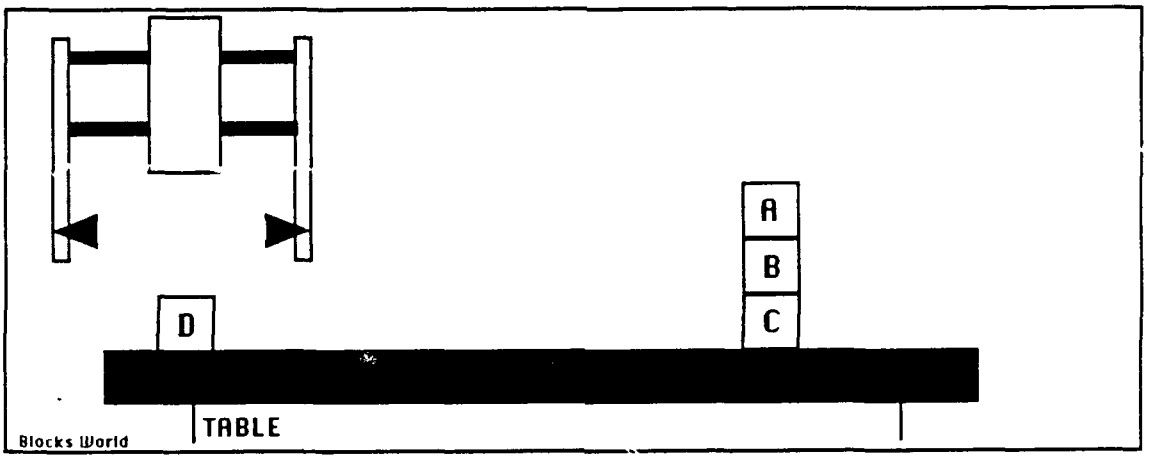

Figure 2.9.2: State of the blocksworld after piling A on B on C.

Put D on B.

```
I understand that you want me to perform put on D and B.


I intend to do
performing put on D and B.

About to do
performing put on D and B.

The act performing put on D and B.
has the following preconditions:
D is held.
B is clear.
They are not satisfied.

Want to achieve D is held.

I deduced
a plan to achieve that D is held is by performing pickup on D.

I intend to do
performing pickup on D.
performing achieve on B is clear.
performing put on D and B.

About to do
performing pickup on D.
```

```
The act performing pickup on D.
has a precondition:D is clear.
It is satisfied.

Now  doing: Pickup D.
Disbelieve D is clear.
Believe D is held.
Disbelieve D is on a table.
I already believe a table is clear.

Want to achieve B is clear.

I deduced
a plan to achieve that B is clear is
  by performing pickup on A and
  then performing put on A and a table.

I intend to do
performing pickup on A.
performing put on A and a table.
performing put on D and B.
```

At this stage block *D* is being held and the system has formed the intention of picking up A. Currently it has no knowledge of the restrictions on the number of arms it has or, for that matter, it has no conception of a hand. Thus it will go on and fulfill its intentions and end up holding more than one block at the same time. This is fine if we allow the system to assume that there is no shortage of hands. However, in the graphics version of the blocksworld we have provided only one arm. Hence our initial description of the blocksworld was not complete with regards to the graphics world (which is where the system is "acting"). Consequently the following incorrect sequence of actions are performed and as shown in the Figure 2.9.3, when the arm goes on to pick up A, D is left hanging.

```
About to do
performing pickup on A.

The act performing pickup on A.
has a precondition:A is clear.
It is satisfied.

Now  doing: Pickup A.
Believe A is held.
Disbelieve A is on B.
Believe B is clear.
Disbelieve A is clear.
```

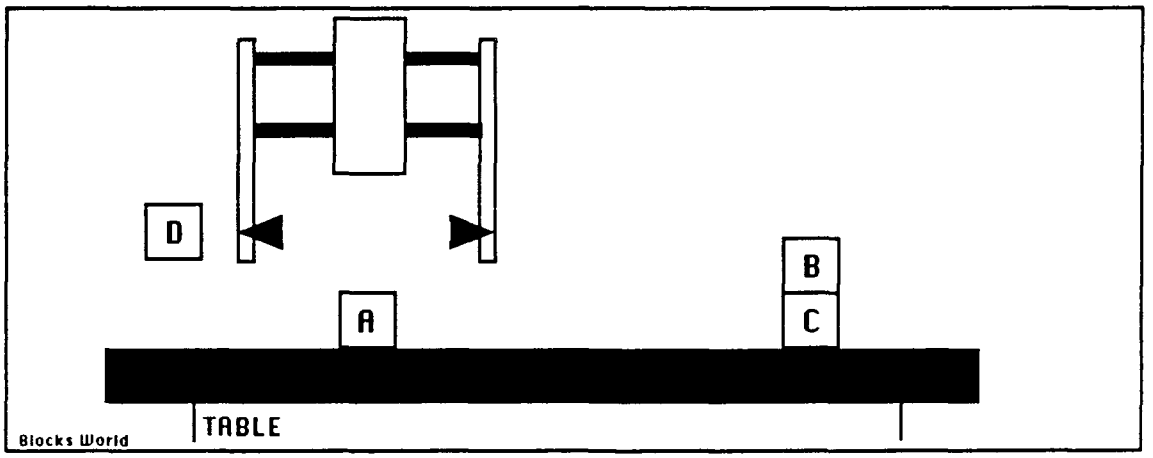

Figure 2.9.3: D is left hanging because the system has no knowledge about the arm itself.

```
I intend to do
performing put on A and a table.
performing put on D and B.

About to do
performing put on A and a table.

The act performing put on A and a table.
has the following preconditions:
A is held.
a table is clear.
They are satisfied.

Now doing: Put A on table.
Disbelieve A is held.
Believe A is clear.
Believe A is on a table.

I intend to do
performing put on D and B.

About to do
performing put on D and B.

The act performing put on D and B.
has the following preconditions:
D is held.
```

```
B is clear.
They are satisfied.

Now doing: Put D on B.
Believe D is clear.
Disbelieve D is held.
Believe D is on B.
Disbelieve B is clear.
```

We can easily correct the above situation by completing the description of the blocksworld by telling the system about the hand and additional information it needs about it. At this point we tell the following to the system:

There is a hand. The hand is empty. Before picking up a block the hand must be empty. After picking up a block the hand is not empty. After putting a block on a support the hand is empty. If a block is held then a plan to achieve that the hand is empty is to put the block on the table.

```
I understand that B2 is a hand.  I understand that a hand is empty.
I understand that before performing pickup on a block,
a hand must be empty.
I understand that after performing pickup on a block,
a hand is not empty.
I understand that after performing put on a block and a support,
a hand is empty.
I understand that if a block is held then
a plan to achieve that a hand is empty is
by performing put on the block and a table.
```

Now that it knows about the hand and related knowledge, we can ask the system to perform a similar task as above and see if it has learned the above paragraph.

Put A on B.

```
I understand that you want me to perform put on A and B.

I intend to do
performing put on A and B.

About to do
performing put on A and B.

The act performing put on A and B.
has the following preconditions:
A is held.
```

```
B is clear.
They are not satisfied.

Want to achieve A is held.

I deduced
a plan to achieve that A is held is by performing pickup on A.

I intend to do
performing pickup on A.
performing achieve on B is clear.
performing put on A and B.

About to do
performing pickup on A.

The act performing pickup on A.
has the following preconditions:
a hand is empty.
A is clear.
They are satisfied.

Now  doing: Pickup A.
Believe A is held.
Disbelieve A is on a table.
Disbelieve a hand is empty.
I already believe a table is clear.
Disbelieve A is clear.

Want to achieve B is clear.

I deduced
a plan to achieve that B is clear is
  by performing pickup on D and
  then performing put on D and a table.

I intend to do
performing pickup on D.
performing put on D and a table.
performing put on A and B.

About to do
```

```
performing pickup on D.
```

We are now at a similar situation as above. The arm is holding A and it has formed an intention to pick up D. However, it has learnt new information since then that it can now use as shown below:

```
The act performing pickup on D.
has the following preconditions:
D is clear.
a hand is empty.
They are not satisfied.

Want to achieve D is clear.
Already Achieved.

Want to achieve a hand is empty.

I deduced
a plan to achieve that a hand is empty is
  by performing put on A and a table.

I intend to do
performing put on A and a table.
performing pickup on D.
performing put on D and a table.
performing put on A and B.

About to do
performing put on A and a table.

The act performing put on A and a table.
has the following preconditions:
A is held.
a table is clear.
They are satisfied.

Now doing: Put A on table.
Disbelieve A is held.
Believe a hand is empty.
Believe A is clear.
Believe A is on a table.

I intend to do
performing pickup on D.
```

```
performing put on D and a table.
performing put on A and B.

About to do
performing pickup on D.

The act performing pickup on D.
has the following preconditions:
D is clear.
a hand is empty.
They are satisfied.

Now  doing: Pickup D.
Disbelieve D is clear.
Believe D is held.
Disbelieve D is on B.
Disbelieve a hand is empty.
Believe B is clear.

I intend to do
performing put on D and a table.
performing put on A and B.

About to do
performing put on D and a table.

The act performing put on D and a table.
has the following preconditions:
D is held.
a table is clear.
They are satisfied.

Now doing: Put D on table.
Believe D is clear.
Disbelieve D is held.
Believe D is on a table.
Believe a hand is empty.

I intend to do
performing put on A and B.

About to do
```

```
performing put on A and B.

The act performing put on A and B.
has the following preconditions:
A is held.
B is clear.
They are not satisfied.

Want to achieve A is held.

I deduced
a plan to achieve that A is held is by performing pickup on A.

I intend to do
performing pickup on A.
performing achieve on B is clear.
performing put on A and B.

About to do
performing pickup on A.

The act performing pickup on A.
has the following preconditions:
a hand is empty.
A is clear.
They are satisfied.

Now  doing: Pickup A.
Believe A is held.
Disbelieve A is on a table.
Disbelieve a hand is empty.
I already believe a table is clear.
Disbelieve A is clear.

Want to achieve B is clear.
Already Achieved.

I intend to do
performing put on A and B.

About to do
performing put on A and B.
```

```
The act performing put on A and B.
has the following preconditions:
A is held.
B is clear.
They are satisfied.

Now doing: Put A on B.
Disbelieve A is held.
Believe A is on B.
Believe a hand is empty.
Believe A is clear.
Disbelieve B is clear.
```

2.10 PLAN RECOGNITION

Our representations for plans and acts also facilitate plan recognition. We have implemented a system which allows the deduction of a set of plans some agent might be performing from information about the acts that agent has been performing. This plan recognition system has mainly been applied to a simple version of the Blocksworld, but was also used for the tutoring domain in order to demonstrate how domain knowledge can be used to narrow the number of possible plans some agent might be engaged in.

In order to explore the advantages and disadvantages of using node based inference and path based inference, corresponding rules were implemented and tested in the Blocksworld. The plan recognition system which uses node based inference was able to deal with complex acts and subplans, *i.e.,* it can identify a plan even if the reported acts are only implicitly represented in complex acts for which there are separate plan-act rules which contain the reported acts explicitly. The problem with node based inference for plan recognition is that it generates a lot of nodes in order to make the component relations explicit, and it cannot deal with uninstantiated plan-rules due to the current implementation of quantified variables.

Using nested entailments, a left recursive representation for plans, and a special representation for the result of a plan recognition process, it was possible to use mainly path-based inference, which increases efficiency and avoids some of the problems related to the quantification of variables in plan-rules. The plan recognition rules were tested successfully for instantiated and uninstantiated plan-rules and for plan-rules with complex acts and corresponding subplans.

Assuming the plan recognition system described in this paper is part of SNACTor's mind, the plan recognition process can be described as follows: SNACTor is told that a third agent is performing certain acts. Using its knowledge base of plan-act and plan-goal propositions SNACTor identifies those plans which contain the reported acts in the correct temporal order and concludes that the agent might perform the corresponding acts or might try to achieve the corresponding goals.

To demonstrate the plan recognition process in SNePS consider the following plan-act proposition: A plan for piling a block on another block on a third block is to first achieve that the first block is on the table and then achieve that the second block is on the first block and finally achieve that the third block is on the second block. This proposition is represented by the following SNePS structure:

```
(assert
  forall ($block1 $block2 $block3)
    &ant ((build member *block1 class block)
          (build member *block2 class block)
          (build member *block3 class block))
     cq (build
          act (build action  pile
                     object1 *block3
                     object2 *block2
                     object3 *block1)
          plan (build
                 action  (build lex snsequence)
                 object2 (build action  achieve
                               object1 (build rel  (build lex on)
                                              arg1 *block3
                                              arg2 *block2))
                 object1 (build
                           action (build lex snsequence)
                           object2
                           (build action  achieve
                                  object1 (build rel  (build lex on)
                                                 arg1 *block2
                                                 arg2 *block1))
                           object1
                           (build action achieve
                                  object1
                                  (build rel  (build lex on)
                                         arg1 *block1
                                         arg2 table))))))
```

Since acts whose action is *achieve* are complex acts in this example, there will be other plan-act propositions which express the decomposition of those complex acts into primitive acts. *E.g.*,

```
(assert
  act  (build action achieve
              object1 (build rel  (build lex on)
                             arg1 blockb
                             arg2 blockc))
  plan (build action  (build lex snsequence)
              object2 (build action  (build lex put)
                             object1 blockb
                             object2 blockc)
```

```
object1 (build action  (build lex pickup)
                       object1 blockb)))
```

The system has knowledge of several deduction rules, which are used to recognize plans. *E.g.*, the following rule says: if an agent performs a reported act, and that act is a component of a plan for some other act, then presumably the agent is performing the second act as well:

```
(assert
  forall ($agent $reported-act)
  ant    (build agent *agent
                act *reported-act)
  cq     (build forall $planned-act
                ant    (build plan-component *reported-act
                              act *planned-act)
                cq     (build presumably (build agent *agent
                                                act *planned-act)))))
```

where plan-component is a virtual arc defined as follows:

```
(define-path plan-component
             (domain-restrict
               ((compose action lex) snsequence)
               (or (compose plan (kstar object1) object2)
                   (compose plan (kstar object1)))))
```

In order to deal with complex acts and subplans, however, more deduction rules are necessary. *E.g.*, the following rules says:

If an agent performs a reported act and if the reported act is a component of a complex act and this complex act is component of a third act then the agent presumably performs the third act:

```
(assert
  forall ($agent $reported-act)
  ant    (build agent *agent
                act   *reported-act)
  cq     (build forall ($complex-act $planned-act)
                &ant   ((build plan-component *complex-act
                               act *planned-act)
                         (build plan-component *reported-act
                               act *complex-act))
                cq     (build presumably
                              (build agent *agent
                                     act   *planned-act))))
```

Assuming that SNACTor is now told that agent John picks up block B, SNACTor can use these deduction rules to answer questions about what acts John might be performing. Here is the reported act:

```
(add agent john
     act   (build action  (build lex pickup)
                 object1 blockb))
```

Now we can ask SNACTor what acts John is presumably performing:

```
(deduce presumably (build agent john
                          act   $planned-act))
```

SNACTor reponds with the conclusion that John presumably is achieving that block B is on block C and that John presumably is piling A on B on C or B on C on A:

```
(M40! (PRESUMABLY
        (M39 (AGENT JOHN)
             (ACT   (M16 (ACTION  ACHIEVE)
                         (OBJECT1 (M15 (ARG1 BLOCKB)
                                       (ARG2 BLOCKC)
                                       (REL  (M13 (LEX ON)))))))))))

(M120! (PRESUMABLY (M119 (AGENT JOHN)
                         (ACT (M45 (ACTION PILE)
                                   (OBJECT1 BLOCKA)
                                   (OBJECT2 BLOCKB)
                                   (OBJECT3 BLOCKC))))))

(M122! (PRESUMABLY (M121 (AGENT JOHN)
                         (ACT (M73 (ACTION PILE)
                                   (OBJECT1 BLOCKB)
                                   (OBJECT2 BLOCKC)
                                   (OBJECT3 BLOCKA))))))
```

If an agent is reported to perform two or more acts in sequence the following deduction rule is needed: if the agent performs two acts *reported-act1* and *reported-act2* in sequence and if the plan for a third act *planned-act* contains two acts *complex-act1* and *complex-act2*, where *complex-act1* occurs before *complex-act2*, and if *reported-act1* is a component of *complex-act1* and *reported-act2* is a component of *complex-act2*, then the agent presumably is performing the act *planned-act*. Here is the SNePS representation of this rule:

```
(assert
  forall ($agent $reported-act1 $reported-act2)
  ant    (build agent *agent
                act   (build action  (build lex snsequence)
                             object1 *reported-act1
                             object2 *reported-act2))
  cq     (build forall ($complex-act1 $complex-act2 $planned-act)
                &ant   (build plan-component
                              (build action  (build lex snsequence)
                                     object1 *complex-act1
                                     object2 *complex-act2)
                              act *planned-act)
                cq     (build &ant ((build plan-component *reported-act1
                                           act *complex-act1)
                                    (build plan-component *reported-act2
                                           act *complex-act2))
                              cq   (build
                                     presumably
                                     (build agent *agent
                                            act   *planned-act)))))
```

Where *object*1 is defined as virtual arc as follows:

```
(define-path object1
            (or (kplus object1)
                (compose (kplus object1) object2)))
```

*object*2 does not need a path definition since plans are assumed to have strictly left recursive representations.

Using the rules and definitions above, it is possible to recognize plans even if the reported acts are not adjacent in the plans, *i.e.*, other acts may occur between two reported acts.

2.11 TUTORING AS PLANNING

One of the original intents of this project was to use plans to understand tutoring paragraphs. It was felt that Dr. Woolf's tutoring expertise, the extensive work at SUNY Buffalo on SNePS, and the availability at UMass of a planning system, GRAPPLE [11], would provide many of the pieces necessary for putting together a system to read and understand paragraphs describing tutoring sessions.

Our first several months on this project were spent in an effort to reconcile these three formalisms. At that time, there was no working SNePS system, so we concentrated on designing a way to use GRAPPLE's plans to understand tutoring paragraphs.[1] This effort resulted in our learning several things, discussed in detail in the rest of this section.

- Reading a story about people using plans is not the same as watching people use plans (or being told how to use plans). GRAPPLE was designed for the latter, but we are working on the former.

- Natural language input contains information (for example, tense, clausal connectives, discourse focussing clues) that requires a recognition mechanism designed to capture and represent it.

- Tutoring is a complex human activity, not amenable to representation by traditional planning methodologies. We felt, however, that of all the planning paradigms available, one closer to that called "reactive planning" best captures the way in which a tutor works, since after each action, the tutor typically monitors the response of the student and may consult other aspects of the current context.

- Recognizing a tutor's goals, through either direct observation or reading paragraphs, is not furthered by recognizing, for example, that she asked a question. Instead, understanding depends heavily on rich knowledge of both tutoring strategies and the subject matter being taught.

How GRAPPLE Might be Integrated into TOM

Plan-based Natural Language Understanding usually takes a story as "understood" if it has matched a story with a known plan or script so that it can infer actions and subgoals that were not explicitly mentioned in the text. We had wanted to use GRAPPLE to match the events in the input paragraph with a known plan. Clearly, GRAPPLE would have been most useful had it been able to narrow its set of hypotheses to a single plan, and allow us to recover an "instantiated plan," one that shows the order and hierarchy of accomplished subgoals and what subgoals were matched with what actions. Unfortunately, GRAPPLE

[1] It should be noted that the version of GRAPPLE we examined in 1987 is perhaps quite different from the current system.

suffered two deficiencies in this regard: abandoned plans and erroneous hypotheses that were never purged; and redundant plans, one for each successfully matched step in the plan. Additionally, GRAPPLE did not keep track of the *order* in which subgoals were achieved; for its purposes, it was merely sufficient to know that they all were.

For tutoring, we would need abstracted plans that represent either *choice points* or *loops* or both. As an example of the former, we wanted an abstract plan to cover a topic which includes questioning the student, and *depending on the answer*, teaching the topic or moving on to something different. As an example of the latter, we wanted an abstract plan for presenting (as in a lecture) a series of examples, without hardwiring the number of examples into the plan.

We came to the conclusion that we would not be able to rely solely on GRAPPLE output for our understanding task, but would also, in some sense, need to save the parse. "Saving the parse" means that we would be holding on to more information from our processing of the input paragraph than the GRAPPLE plan(s) and the obvious instantiations of plan variables such as "Tom" for "student." In general, we would need to save information that GRAPPLE had no interest in and would throw away. More specifically, we came up with two independent reasons for saving the parse.

First is in the event that GRAPPLE cannot provide us with a single plan, and thus, presumably, returns us a set of plans. We would still want to be able to answer questions about what went on: we would want to be able to give a full description of the actions mentioned, even if we could not characterize the strategies (plans) behind them. It would be very difficult and tedious to try to recover this description from a bunch of partially instantiated GRAPPLE plans.

Second, there are times when the paragraph will describe all the actions that are to be matched by GRAPPLE, but in a different order from their occurrence. For example, in our tutoring paragraph, the first few sentences could be restated thus:

> Nancy asked Tom if an inanimate object exerts a force. Nancy pointed to a book on the table. She asked if the table exerts an upward force on the book. Tom said no to both questions.

The order of occurrence of events is essential to plan understanding, and GRAPPLE itself has no way to reorder steps, because it is designed for "real time" plan recognition. Therefore, we would need to reorder as necessary in our own implementation. Assuming we would be feeding GRAPPLE information after processing each sentence, we would be faced with the possibility of having to backtrack in order to correctly order the steps. Alternatively, we could hold onto all the results of the parse, do a reordering pass, and hand the results to GRAPPLE in the correct order.

The first option would allow us to work incrementally, but would become wasteful when backtracking occurred. The second option avoids wasted work, but disallows the possibility of using GRAPPLE to form any predictions about upcoming input, which could conceivably help the parsing process.

We found neither of these options attractive, and this situation, in addition to GRAPPLE's own problems, persuaded us that GRAPPLE would be of limited benefit to our

project. We therefore decided not to use it and turned our energies toward finding a better-suited planning formalism.

Condition-Action Pairs

Our study of the tutoring paragraph convinced us that to understand it our system need rely only minimally on plans. A tutor may incorporate overarching strategies into her tutoring, such as "examples are good" and "present easier material before harder," but when she decides what action to take next in teaching a student, she typically examines the current context rather than consulting some tutoring schema. This context includes what she has already taught, what she believes the student understands, what the student's most recent response was, what material is yet available to teach on the topic (*domain knowledge*) , and what *tutoring strategies* may apply for teaching it.

Domain knowledge of Physics, for example, includes information that relates the concept of "inanimate object exerting a force" to the concrete example "table exerting an upward force on a book." The concept is also related to the *anchor example* (one that a student is expected to understand because it involves his direct experience) of "hand exerting an upward force on a book." The availability of such examples helps guide the tutoring process.

Tutoring strategies can best be conceived of as *Condition-Action pairs*: for any tutoring situation or set of conditions, any number of actions may be appropriate; conversely, an action may be appropriate to any number of situations. A tutor will choose an action based on the conditions. A system trying to understand a tutor's activity will match the tutor's action and attempt to match a corresponding set of conditions. It is in this weak sense that plans are used in our system. Figure 2.11.1 gives some sample conditions and actions.

We believe that the planning formalism being developed by Dr. Shapiro and his group at SUNY Buffalo are adequate to represent these Condition-Action pairs. Using his paradigm, the Condition-Action knowledge is implicitly incorporated in our inference rules.

CONDITION	ACTION
student doesn't understand topic	give anchor example (found in KB)
tutor wants to bridge	repeat question go back to an earlier concept give anchor example (found in KB)
tutor wants to introduce topic	ask question about topic
tutor wants to test student's knowledge	give concrete example of topic
tutor wants to review a previous topic	tell student to explain topic

Figure 2.11.1: Sanple Condition-Action pairs.

2.12 UNDERSTANDING SITUATIONS

In this section, we discuss the definition and processing of words such as "situation," "example," and "case." We have concluded that a *Situation*, the referent of this word, is an aggregate of model objects picked out by a *Situation Index*. Situations are states of affairs characterized by concreteness and tension. We show further that context must inform the comprehension of this difficult lexical item.

Words such as "situation" are important because they occur frequently in natural discourse, and they are interesting because they are like anaphoric terms ("he," "it") and deictic terms ("this," "that"), but they have a much richer semantics. Consider the following usage:

> Penni and Scott are writing an IJCAI paper. Their first draft was shot down by their fellow researchers. In this situation, they squabbled more than usual.

Most people would agree that this paragraph contains a description of a situation, and more importantly, that the lexical item "situation" successfully refers to it. However, understanding the exact reference of the word "situation" is complicated by three factors:

1. "Situation" places significant, but ill-defined, constraints on its possible referents. This example shows elements of concreteness (particular people writing a particular paper), states (a bad first draft), and change (such as increased squabbling). These are much more complex constraints than the simple number, person, and gender features of pronouns.

2. "Situation" is nevertheless a weak descriptor, because situations come in so many different kinds—different sets of entities can be referred to as a situation, depending on the context.

3. Situations might not exist *a priori*, but instead be created *a posteriori*. That is, the set of objects that make up the situation may not previously have been aggregated. The objects aggregated by "situation" reside in the Model,[1] which is constructed in the process of understanding the text.

This section attempts to define the semantics and processing of situations.

[1]This Model is informed by, but is distinct from, our domain models, which contain world knowledge of various possible domains of discourse. The Model may contain explicit structural or lexical information; this needs to be saved for some understanding tasks (for example, Webber's discourse anaphora (1987)), but it is not necessary for processing situations.

Types of Reference

Model Anaphora, which we define to be the determination of the reference of words such as "situation," naturally extends other work on anaphora. Many researchers have studied pronoun anaphora (coreference between a noun phrase and a personal pronoun): Hobbs (1978), Grosz and Sidner (1986), [8], and [35]. Work on temporal anaphora (coreference between temporal expressions) has been done by [21] and [9], among others. Webber has also worked on the reference of demonstratives such as "this" and "that"; she has termed this variously as "event reference" [36] and "discourse anaphora" [37]. The 1987 paper also introduced the notion of *individuating reference*, which plays a key role in our Model Anaphora.

Text comprehension involves *reference*, which we define as the link from an expression in the text (for example, a noun phrase or a proposition) to a Model Object,[2] called the *referent*. The arrows in Figure 2.12.1 are examples of such reference links.

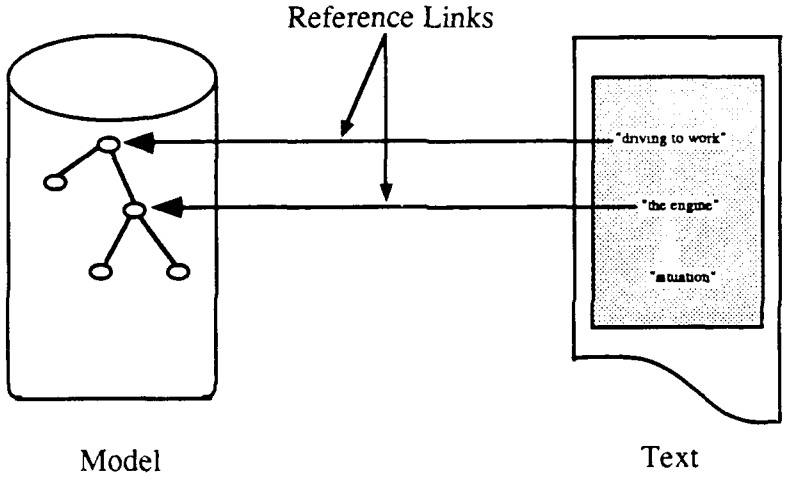

Figure 2.12.1: the reference links from the text to the Model

Sometimes, the referent of an expression is a *set* of things in the Model. Such a set acts as an *individual*, in that it can be referred to, can have qualities attributed to it, and can participate in relations. [36] calls a reference that creates a new individual an *individuating reference*, depicted in Figure 2.12.2. One can also view this as creating a new first-class object, since only first-class objects can be referred to and predicated.

Not all aggregate model objects are the result of individuating reference: some aggregates already exist as part of a domain model. For instance, the story of Romeo and Juliet is clearly an aggregate, being composed of many people and events, but it is easily available as a referent in discourse without being created anew.

[2]A Model Object is any first-class object in the Model (essentially, a first-class object is anything that can be pointed to). Therefore, Model Objects include relations, events, states, and so forth, as well as physical objects. Recall that the Model records our understanding of the text, and is distinct from domain models.

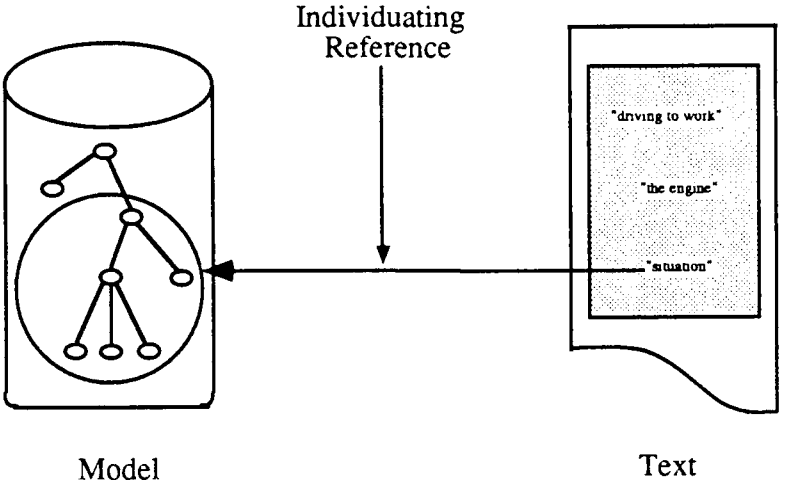

Figure 2.12.2: An expression referring to a set of objects as though it were an individual. The reference that creates this set is called an *individuating reference.*

We believe that a situation is represented as an aggregate model object and that the word "situation" is often used as an individuating reference. There are two issues to address: defining just what can be termed a situation, and processing a reference to a situation (which may involve an individuating reference). The next section deals with the problem of a proper definition of this term, and Section 2.12 deals with processing.

The Definition of Situations

We have identified several factors which influence whether an expression describes a situation. Consider examples (1) through (4) below. When asked to judge which examples can be characterized as situations, speakers generally agree that (4) can be a situation, but (1) cannot. However, there is less agreement on whether the intermediate expressions may refer to situations.

(1) running
(2) running the Boston Marathon
(3a) James running the Boston Marathon
(3b) James having run the Boston Marathon
(4) James having run the Boston Marathon last April

It is possible to imagine contexts in which any of these phrases describes a situation, but in some cases, the contexts are more natural and more available than in others. We believe that these expressions differ by their *concreteness.*

Concreteness is characterized in the psychological literature by imageability (for example, [20]). In our examples, the increasing specificity supplies the details that aid in building

up a mental image. Specificity, however, is not the right measure of a situation. Consider example (5):

(5a) playing a violin
(5b) playing a Stradivarius

Most people judge these as equally good (or bad) as situations, yet (5b) is clearly more specific, though it is no more concrete (imageable). This is similar to [22] notion of *basic level categories*; one formulation of a basic level is that it is the highest level for which the *prototype* is imageable (hence, concrete). Thus, concepts at or below the basic level are equally concrete, and therefore their concreteness equally affects judgements of whether something is a situation.

Intuitively, a situation is a state of affairs—a description of the world (or some small part of it) at some point in time. Note that the questions "What is the situation with X?" and "What is the state of X?" will elicit similar, if not identical, descriptions. Certainly, the description may combine present and past actions, especially those that are related (causally or otherwise) to the current state of affairs:

> The situation is that I dialed this long-distance number and it's been over thirty seconds and it still hasn't rung. Strange.

Clearly, this situation includes the past action of dialing the phone. On the other hand, a situation need not mention past actions. For example, the situation at the beginning of *Romeo and Juliet* can adequately be described as the feud between the Montagues and the Capulets, without mentioning any particular actions.

A situation usually contains *tension*, which we take as *potential for change*. In Example (6) below, (6a) is a better situation than (6b) because there is a greater potential for change. Most people consider (6b) too uninteresting to be a situation at all. Of course, in certain contexts, (6b) could be considered a situation: a textbook on erosion might discuss the changes in that situation.

(6a) A boulder teetering on the edge of a cliff.
(6b) A boulder sitting in the middle of a plain.

Example (6) also demonstrates that tension need not result from agents and their goals, as it does in the Romeo and Juliet situation.

We conclude that situations are states of affairs characterized by concreteness and tension. Concreteness reflects how easily people can construct a mental image of the situation. Tension reflects people's judgements of the potential for change.[3] Obviously, judgements of these characteristics will vary from person to person; nevertheless, they put significant constraints on what can be a situation.

[3]Some situations, unfortunately, seem to be ones in which people nevertheless judge there to be little potential for change. For example, consider the situation of being trapped in the center of a row at a boring lecture.

Processing

Situations are commonly referenced with phrases such as "the situation with X" or "the X situation." Sometimes, depending on X, more specific phrases are used, as with "the situation in the Middle East," which is preferable to "the situation with the Middle East." These modifying phrases include the *Situation Index* for each of these Situations. A *Situation Index* is a key that picks the situation out of the Model, distinguishing foreground from background, so to speak.

The situation index is usually some common or unifying aspect of the situation. Thus, if we speak of "the IJCAI paper situation," (see our first example), the situation index unites the situation. This situation is also successfully referred to by "the situation with Penni." Indeed, the situation index need not be explicitly mentioned in the text. For example:

> I was driving to work with the carpool, as usual, and the engine threw a rod. It's shot, so now I've got to get a new one. That's the situation with my car.

This reference is successful because the mention of "driving" and "engine" bring "car" into *focus* [32]. Furthermore, replacing the referring phrase with "the situation with the carpool" does not as easily refer to the car situation. Therefore, we conclude that, to a first approximation, it is elements in focus, regardless of explicit mention in the text, that are available as situation indexes. We have found that the *topic* of the paragraph is usually what is in focus by the end of the second sentence, and the topic is the most likely source of the situation index.[4]

We take references that make explicit mention of the situation index to be the canonical case; we process other references by first determining the situation index, thereby reducing them to the canonical case. Thus, Model Anaphora is a two-step process:

1. **Determine the situation index.** Where this is explicitly mentioned, the determination is trivial. Otherwise, the situation index must be computed from the context.

2. **Using the situation index, search the Model and look for matches.** Since we take as a departure point the actual result of parsing the text, the search will be through a subspace of the Model. The matching Model Objects will yield a set of candidates, some of which will be discarded because they are not states of affairs or because they lack concreteness or tension. If this does not reduce the set of candidates to a single situation, we choose the most recent candidate.

Our current research concerns the comprehension of the following paragraph, so we will use it to illustrate Model Anaphora:

> Nancy asked Tom if an inanimate object exerts a force. Tom said no. Nancy pointed to a book on the table. She asked if the table exerts an upward force on the book. Tom said no. Nancy placed the book on Tom's hand. Tom exerted a great force to keep the book steady. Nancy asked Tom to compare the two situations. Tom said the table exerted a force on the book.

[4] See Section 2.15 for an alternative method of finding the paragraph's topic.

Nancy sets the topic of the paragraph with her first question to Tom. This marks "inanimate object exerts a force" as the topic; the Model Objects *inanimate object*, *exerts*, and a *force* are available as situation indexes. Because of our domain knowledge of physics, we know that her second question, which is more specific, is subsumed by her first, and does not change the topic. When Nancy refers to "the two situations" without supplying a situation index, either "inanimate object" or "force" is probably the index she has in mind.

Supposing we take "force" as the situation index, we search in the Model for Model Objects that match it. We find the first situation because "an upward force " is a kind of force and Nancy has asked Tom whether the books exert an upward force on the table. Hence, *the books on the table* is a situation. The second situation is found again because force is explicitly mentioned, hence *the books on Tom's hand* is also a situation.[5] Although force is mentioned in the first sentence, "if an inanimate object exerts a force," this is discarded as a situation because it lacks concreteness.

Summary

We have outlined the two phases of processing Model Anaphora and have shown how we can find the referents for "situation" in an sample text. The first step in the process is finding a situation index, which we have defined as the key that picks the situation out of the Model. The second step involves searching the Model for occurrences of the situation index and determining whether any of these is part of a situation.

In conclusion, we claim that situations are states of affairs that exhibit concreteness and tension, and that the process of referring to them involves a situation index. We believe that Model Anaphora is a phenomenon common to a large class of nouns, including "case," "disaster," and "example." These nouns differ somewhat in their semantics, but they all make individuating references to sets of Model Objects.

[5]Both situations are larger than we have described them here. For instance, Nancy would include in her representations the forces involved in the two situations. Tom will, too, once he understands the concept.

2.13 IMPLEMENTATION OF THE TUTORING OBSERVATION MODULE

Our program, TOM (Tutoring Observation Module), reads a paragraph, using the SNePS Augmented Transition Network (ATN) to parse the sentences and represent their meaning with SNePS nodes, which are automatically integrated into the SNePS network. Next, the Sidner Focussing Algorithm is executed. This algorithm tracks the focus movement through the paragraph, that is, as the focus changes from some element of the previous sentence to some element of the current sentence. The focussing algorithm was originally designed to work sentence by sentence and, although we considered implementing more dynamic focussing, we decided to stay with sentence-based processing. This required us to make the focussing algorithm a separate stage after the parser, since the ATN might backtrack at any time. Because paragraphs in the tutoring domain will contain many indirect questions, we decided not to trigger question-answering within the ATN. Instead, we look at the topmost node that represents the meaning of the current sentence and, if it's a question, we invoke the inference engine to deduce the answer. When this is completed, the answer is printed. Since the research is not concerned with Natural Language Generation, the answer is printed simply by looking up a sentence corresponding to the answer that was deduced. This technique is obviously deficient in the long run, but it enabled us to concentrate on the primary task of text comprehension. We believe that the SNePS generator will eventually be able to handle our generation demands. An outline of TOM's organization appears in Figure 2.13.1.

Implementing the Grammar

The paragraph that we have studied most extensively and that TOM can now read contains very natural English, and that presented some problems. Some phenomena are simply outside the scope of this research and are among the open problems in AI and NLP today. For

```
Loop forever
   Read a sentence;
   Call ATN to parse it;
   Call Focussing Algorithm to update focus;
   If sentence is a question then
      Deduce answer;
      Print corresponding sentence
   end if
end loop
```

Figure 2.13.1: an outline of TOM's organization

example, the original paragraph made great use of modals: "Tom had to exert..." rather than "Tom exerted...." To properly represent these meanings would demand representing possible worlds and intentions of agents, which would have been extremely complex. Therefore, we simply rewrote these sentences in simpler language.

The first sentence originally read:

1. Nancy asked Tom if an inanimate object, such as a table, can exert a force.

The parenthetical expression "such as a table" presented unexpected difficulties. It could be parsed, but there was much controversy over the meaning of the expression, the representation of that meaning, and the ultimate effect of that meaning. Eventually, we decided that the effect of "such as" phrases is to focus the attention of the human listener or, by analogy, to advise the computer's inference engine so that it can focus its inferences in a particular way. In a way, it's as if the speaker expects that the hearer will attempt to answer the question by using a case-based approach: examining cases of inanimate objects and deciding if any of them can exert a force. The "such as" phrase prompts particular kinds of cases to be examined. Consider asking a naive student whether a hammer or an anvil can exert a force. The student is likely to say that the hammer can and the anvil cannot, simply because the hammer is active and the anvil passive. Thus, the "such as" expression focusses the hearer's attention on passive inanimate objects, without articulating that concept. Once we agreed that this is the correct view of the "such as" phrase, we also knew that it could have no effect on the SNePS inference engine, since the SNePS inference engine is not case-based and has no attentional mechanisms. Therefore, we dropped the phrase; we would be parsing it to no purpose.

The second sentence is both very simple and very interesting.

2a. Tom said no.

Actually, the original sentence was

2b. Tom said he didn't think so.

We changed it because we considered the latter simply a needlessly complicated idiomatic expression with essentially the same semantics as the former. The most important difference in semantics is that the latter expression conveys Tom's doubt is his belief—a difference we could not have accounted for, since the SNePS inference engine is a symbolic logical system, not a probabilistic system with degrees of belief. Still, "Tom said no" is more than interesting enough, because the key problem is determining what he said no *about*. He is negating some proposition, but what proposition? We call this problem *Propositional Anaphora*.[1] Properly understanding this sentence requires awareness of the continuity of sentences at the discourse level, higher than the sentence level. In Section 2.13 we discuss how we use an extension to Candy Sidner's focussing algorithm to handle Propositional Anaphora.

[1]The same issue arises, of course, with affirmative sentences such as "John said yes."

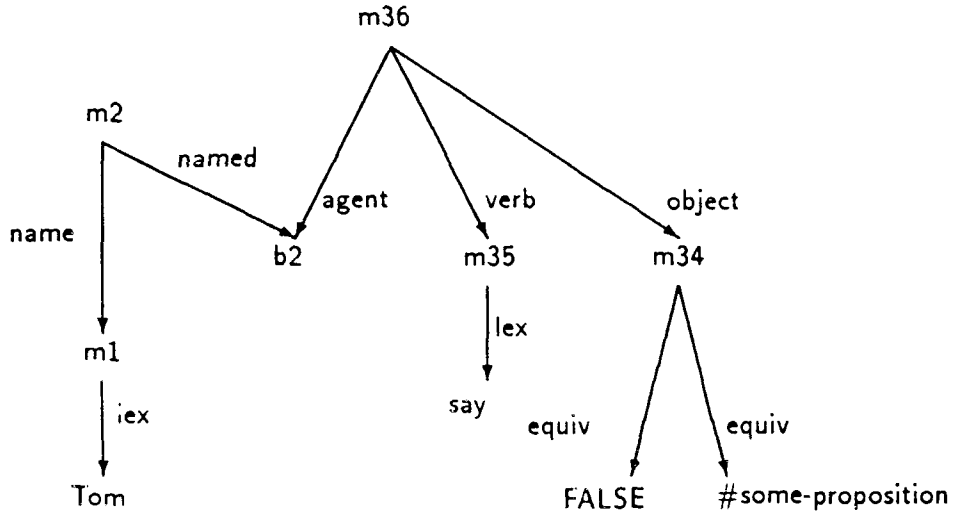

Figure 2.13.2: the parser's representation of the meaning of "Tom said no"

Because the focussing algorithm is executed after the ATN has finished parsing the sentence, the representation of the meaning of "Tom said no" is as depicted in Figure 2.13.2. The "#some-proposition" is just a convenient label for what is an atomic base node, meaningful only by what points to it; for our purposes, it's just a placeholder. So, the network reads "Tom said that <some proposition> is equivalent to False." "False," of course, is a constant in the network. At this point, we don't know what he said was false. Later, after the focussing algorithm runs, we get the network fragment in Figure 2.13.3. You can see the need for the placeholder node, as it is the connection between 'False' and the node M12, which is the focussed element.

The next interesting sentence is the following

7. Tom exerted a great force to keep the book steady.

In current Linguistics literature, the phrase "to keep the book steady" is known as a *rationale clause* [10]. Rationale clauses are deemed to attach directly to the **S** node, rather than to **VP**, as a relative clause would, yet they are like relative clauses in that there is usually a *trace*—a item which is deleted because it is identical to something in the main clause. In this rationale clause, the trace is the *agent*, Tom, who is the one keeping the book steady. This sentence also exhibits a *small clause*, namely the phrase "the book steady." We parse this into the proposition `steady(book)`, which is then the object of "keep." It makes sense for "keep" to take a state (the steadiness of the book) as its object. We do not claim that our ATN can handle all small clauses and rationale clauses, but we have gotten it to handle, in a robust and general way, sentences like this one.

The most difficult sentence we faced was the next one:

8. Nancy asked Tom to compare the two situations.

This has already been discussed at length in Section 2.12. The sentence also posed a slight parsing problem, in that the phrase "to compare the two situations" can be read

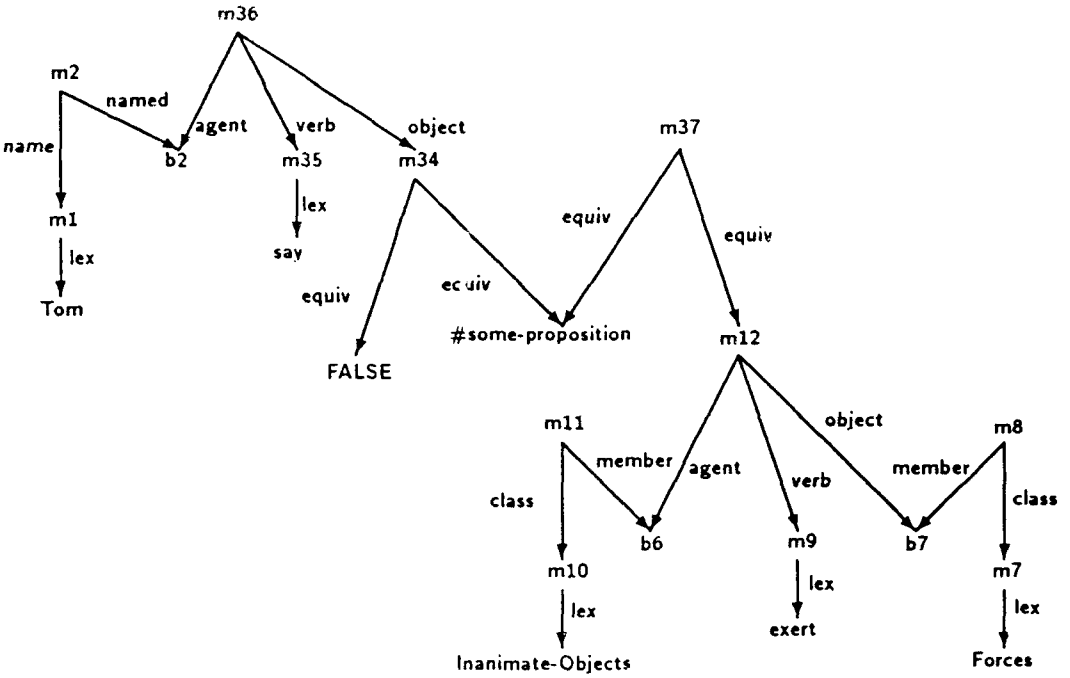

Figure 2.13.3: the representation of the meaning of "Tom said no" after the focussing algorithm runs

as a rationale clause. We were forced to be much more rigorous in annotating verbs with features to say what arguments they can or must have. In this case, we solved the problem by having "ask" require a **VP** as its object. While the ATN and lexicon are somewhat improved in their use of features to control the parsing, this continues to be a weakness of the system. Ideally, the system would have a fairly complete set of subcategorization frames and thematic grids, with a mapping from the former to the latter, and have the subcategorization frames take a much stronger role in the parsing, rather than letting the structure of the ATN control the parsing.

The last grammatical problem we tackled was parsing the questions that follow the paragraph:

Q1. Why did Nancy ask Tom about the book on the table?
Q2. Why did Nancy place the book on Tom's hand?
Q3. Why did she ask him if inanimate objects exert a force?

Previous SNePS implementations had concentrated on "wh-questions," such as "what did Lucy pet," which would, as a side-effect of parsing, cause SNePS to search the network and reply, for example, "Sweet Lucy petted a yellow dog named Rover." Because we believed our domain would present us with indirect questions—statements such as "Nancy asked Tom what the book weighed," we did not want the system to answer *every* question it read. Therefore, we chose a simple representation for questions, and then, *after the parsing*, if the topmost node of the representation of a sentence is a question node (which means it's

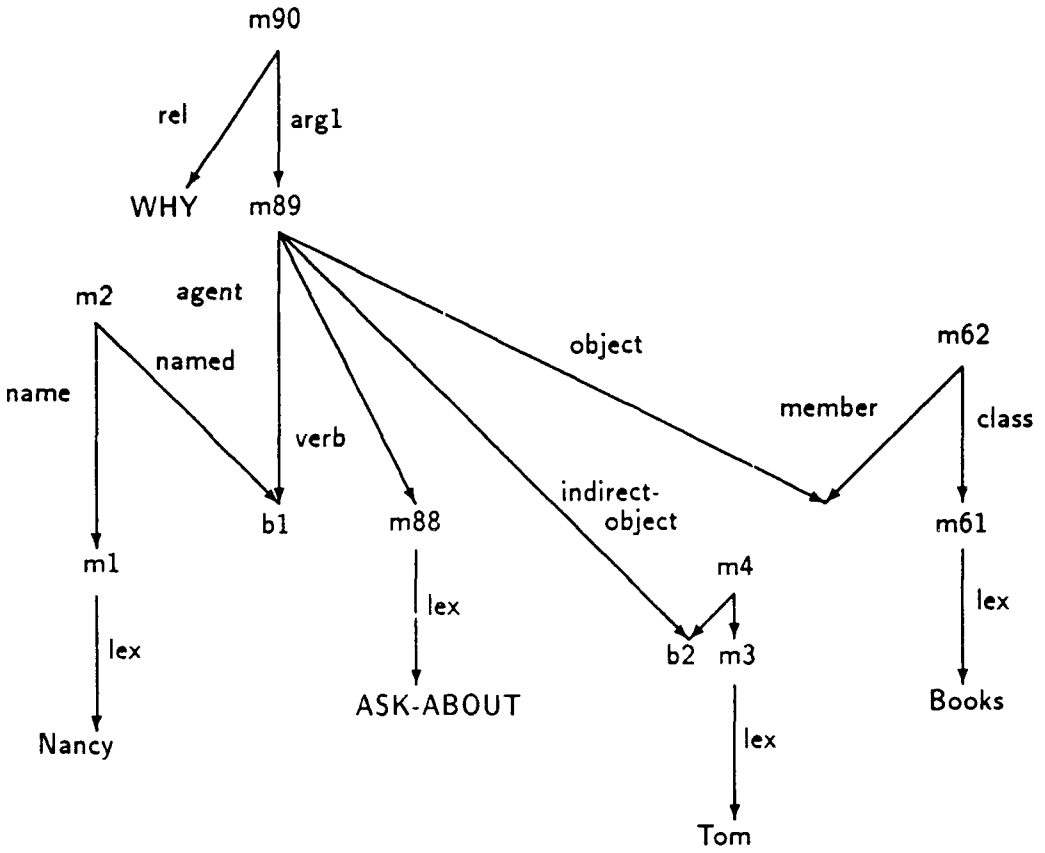

Figure 2.13.4: the representation of the meaning of "Why did Nancy ask Tom about the book on the table?"

a direct question, not an indirect question), we answer the question. Figure 2.13.4 shows the representation for the first question.

Note that the verb in Figure 2.13.4 is "ask-about." We felt that "ask...about" could be treated as a verb-particle combination, rather than a simple verb with a prepositional phrase. Technically, it should be the latter, since "about" does not undergo particle movement (one of the key diagnostics of a true particle construction). The change was motivated primarily on semantic grounds, because we wanted "the book on the table" to be the object of the sentence, so that it would fulfill the requirement that the verb "ask" take an object. (See the discussion of sentence 2.13, above.) Moreover, several other functions also look for the object of the sentence, and this change makes those functions simpler. The problem arises because the system conflates syntactic and semantic roles in the label **object**. While our solution works, it is not sufficiently general, and a uniform distinction between syntactic positions (of subcategorization frames) and thematic roles (of thematic grids) would solve the fundamental problem. Then, at the syntactic level, "the book on the table" can be the object of the preposition "about," and yet be mapped to **theme** at the semantic level, and all semantic functions can uniformly refer to **theme** rather than **object**.

Parsing the questions also required us to substantially rewrite the grammar so that it

could handle the inversion, auxiliary-insertion and other syntactic phenomena of questions in English. The resulting grammar has been tested on a large suite of sentences and is quite robust. The whole ATN grammar is included in appendix 2.A.

Implementing Sidner's Focussing Algorithm

As described in Section 2.12, a major concern of our research has been developing a theory of Model Anaphora. Since we considered this effort an extension of other theories of anaphora resolution, and since, in particular, our theory depended on the notion of focus, we decided to incorporate Candy Sidner's focussing algorithm (1979) into our system. Her work remains one of the cornerstones of focussing theory, and since her algorithm is so well-known, we thought it would be well-understood and easy to customize for our own purposes.

Unfortunately, several difficulties arose from our assumptions. While implementations are reputed to exist, we could not find someone actually able to give us one; Sidner's thesis itself contains only an English version of the algorithm, which is obscure in some places. Further, it became apparent that someone else's implementation would probably not help us too much, since the implementation naturally depends greatly on the structure of both the knowledge representation and the parser. So we eventually decided to implement the algorithm from scratch, using the thesis as a guide. This exercise proved instructive of the pitfalls of such an endeavor.

Sidner's focussing algorithm was motivated by a need to find ways to constrain the search for cospecifications of anaphora in multi-sentential text. An *anaphor* is a lexical item that derives its "meaning," or specification in a knowledge base, through association with some other lexical item. The most common anaphora are definite noun phrases—pronouns and noun phrases (NPs) with definite articles (such as "the"). Such NPs *cospecify* with other NPs in the text.[2] Thus, for example, in the following text "she" cospecifies with "Nancy" and "the book" cospecifies with "a big red book."

> Nancy was leafing through a big red book. She remembered fondly the time she used the book to tutor Tom about physics.

The idea behind focussing is that only one item in the text can be in focus at a time. What is in focus constrains the resolution of anaphora; conversely, use of anaphora signals either continuation of the focus or the movement of the focus to some other item in the text.[3]

The algorithm works by processing one sentence at a time. For each sentence, an ordered list is constructed of all items that may come into focus at some point. For the first two sentences, this is the Default Expected Focus list (DEF); for all subsequent sentences, this

[2]Most researchers refer to this phenomenon as coreference rather than cospecification; Sidner prefers the latter because in a program no referring to the real world is happening, only specification of particular items in a knowledge base.

[3]For a much fuller discussion of these ideas, see [32].

is the Potential Focus list (PFL), which differs from the DEF by not including the agent of the sentence. Each sentence's Alternative Focus List (ALFL) is the DEF or PFL from the previous sentence. The processing of each sentence results in either *confirming* the focus or *moving* it; the ALFL is one set of possible new focuses. If the focus is moved, the old focus is pushed onto the focus stack. The focus stack is another source of new focuses. If the focus is moved to one of these (that is, back to an item that has already been in focus), the items above it in the stack are discarded. There is no focus for the first sentence, since focus is a property of multi-sentential text. Processing for the second sentence, then, is a matter of *establishing* the focus, rather than confirming or moving it.

The focussing algorithm has a companion focussing algorithm for actors that handles the resolution of personal pronouns. We did not implement this because this issue is tangential to our main concerns and a simplistic solution was already available by use of the SNePS Uniqueness Principle. We did not implement the entire focussing algorithm, either, since some of it was not needed for the paragraph we were understanding. Our implementation also includes several extensions for cases not covered in the original, as described below.

The interface between SNePS and the focussing algorithm is not very clean. The algorithm is called after each sentence is parsed, and handed a list of SNePS nodes corresponding to phrases. Unfortunately, the algorithm depends on the syntactic roles of each of the phrases in a sentence as well as the actual linear order in which the phrases appear. The SNePS parser, on the other hand, is only concerned with the semantic roles played by elements of the sentence, and therefore in principle does not save any syntactic information. Hence, the first task of the focussing algorithm is to recover as much of this information as possible. We also had to add to the parser a mechanism for explicitly recording the definiteness of each NP, something to which SNePS is indifferent but which is clearly crucial to the algorithm.

The rest of this section briefly sketches some of the interesting aspects of the algorithm, as it functions in our tutoring paragraph, reproduced here.

> (1) Nancy asked Tom if an inanimate object exerts a force. (2) Tom said no. (3) Nancy pointed to a book on the table. (4) She asked if the table exerts an upward force on the book. (5) Tom said no. (6) Nancy placed the book on Tom's hand. (7) Tom exerted a great force to keep the book steady. (8) Nancy asked Tom to compare the two situations. (9) Tom said the table exerted a force on the book.

Sentence (2) immediately presents us with a type of anaphora not covered in the original algorithm. When Tom says "no," he is negating some proposition, and to understand the sentence we need to find out which proposition this is; thus, we call this Propositional Anaphora. Following the principles of the focussing algorithm, we believe that the proposition being negated will be the one in focus, and indeed, "if an inanimate object exerts a force" is the predicted focus at the second sentence.

This example demonstrates another extension we made to the algorithm: Sidner's version did not account for sentences with subordinate clauses. We decided that subordinate

clauses, including the rationale clause in sentence (7), fill roles similar to those filled by NPs, and so we could treat them equivalently. However, we also felt that the constituents of a subordinate clause should be available to the focussing algorithm as well, so these are also included in the DEF/PFL.

Our most interesting use of the algorithm occurs in sentence (8). Previously, the topic of the paragraph was determined, at the second sentence, to be the element in focus, that is, "an inanimate object exerts a force." The word "situations" triggers a part of the focussing mechanism responsible for resolving such Model Anaphora (see Section 2.12). Any of the elements of the topic, namely *exert*, *inanimate object*, and *force*, may be used as the situation index; the current implementation uses the first of these. "The two situations" is thus found to specify the aggregate of *table exerts an upward force on the book* and *Tom exerts a great force to keep the book steady*. Each of these situations is itself an aggregate of the entire situation and all of its elements.

At this point in the paragraph, during the processing of sentence (8), the focus is the *book* cospecified by "the book" in sentence (7). The focussing algorithm includes a step by which if a definite noun phrase in the current sentence (here, "the two situations") *implicitly cospecifies* the focus, the focus is confirmed. Sidner defines implicit cospecification as a case in which the definite NP specifies a Model Object that is closely related to the specification of the focus. In this example, the *book* is a situation element of one of the two situations; this relationship is close enough to warrant confirmation of the *book* as the focus.

Implementing Sidner's focussing algorithm and making the necessary extensions to it proved to be more work than we had anticipated and in retrospect was probably a poor design decision. The apparent appeal of an "off the shelf" algorithm was offset by the difficulty of engineering the interface with SNePS, as well as the limited coverage of the original algorithm. In fact, we learned the hard way that an off the shelf algorithm is a far cry from off the shelf software, and we would have been better off in this case building a mechanism of our own rather than trying to fit an algorithm to our needs.

Implementing the Knowledge Base

TOM's knowledge of tutoring and physics is represented as inference rules, stored in the SNePS network and executed by the SNePS inference engine. They are triggered by the need to answer questions asked of the system. Calling the inference engine results in new nodes being asserted and returned, whereupon these nodes are mapped to sentences that answer the user's question. This section will review the deductions involved in answering the first question.

The representation of the first question was shown in Figure 2.13.4 (page 75). "Why" questions are taken to ask what goal some action or plan is subserving. Therefore, we turn that question into the following deduction:

```
(deduce plan (build agent            b1
                    verb             ask-about
                    indirect-object  b2
                    object           b9)
       goal v1)
```

The node "V1" is a new variable node, which will be bound to the node which is deduced to be the answer to the question, which is essentially "what is the goal of this plan." The call to DEDUCE initiates the backward-chaining ability of the SNePS inference engine. It creates a pattern node which matches the consequent of rule 1, shown in Figure 2.13.5.

This rule is essentially knowledge about tutoring, for it concerns asking someone about specific examples of a topic in order to probe his knowledge of the topic. Note that the agent, x, which in this case is Nancy, is mentioned several times in the rule: Nancy did the asking, so it's Nancy who wants (Nancy) to know Tom's knowledge of the topic. The use of the agent is a philosophical difference between UMass Amherst and SUNY Buffalo, since the UMass group believes it is important to mention the agent, so that the agent can constrain the inferences and be represented in the answer. The UB group, on the other hand, believes it is important to represent the commonality of different agents' plans and not mentioning the agent in a rule produces more general inferences.

Once the inference engine has matched the consequent of rule 1, it wants to prove the antecedents of that rule. The first antecedent is just the act itself and so is trivially true.[4] The second antecedent is that "B9" is a situation index for some (as yet unknown) situation "e." B9 *is* a situation index (we determined so when processing the Model Anaphora in "Nancy asked Tom to compare the two situations"), and so "e" is bound to the corresponding situation, the node b19, shown in Figure 2.13.6.

Antecedent 3 is that "e" is a situation. This follows straightforwardly from its having situation elements such as B9, so we won't go into this. The fourth antecedent, however, is somewhat complicated, as it determines whether "e" is an example of Statics. That matches the consequent of rule 4, shown in Figure 2.13.7, with "z" unified with "Statics."

This rule captures some of the program's knowledge of Physics, since it defines conditions sufficient for an example of statics. The first antecedent is that "s" (which is our "e" from rule 1) is a situation; we already know this is true. Second, we check that "s" has situation elements "x" and "y." This binds "x" to b9 and "y" to b10. Next, we check that both "x" and "y" are physical objects. The variable "f," in the next antecedent, is open and will be bound to something deduced from yet another rule (one which deduces that if "x" supports "y," then there exists a force, "f," between them). We check that "f" is a force exerted by "x" on "y" and by "y" on "x." This completes the backward chaining through rule 4, and we can now return to rule 1.

[4]Actually, it's not quite so trivial. The act is true because the parser asserted it to be true; that is, the parser takes the statements it reads to be true. One could imagine a more suspicious system that might well answer this question: "I'm not certain that Nancy did ask about the books on the table." This is a minor quibble, as most AI systems believe what they're told, but it's worth mentioning why this proposition is true.

```
(assert
  forall ((new-variable-node x) ... y, z, e, si, k)
  &ant  ((build agent            (node-named x)
                verb             ask-about
                indirect-object  (node-named y)
                object           (node-named si))
         (build arg1 (node-named si)
                rel  situation-index
                arg2 (node-named e))
         (build arg1 (node-named e)
                rel  situation)
         (build arg1 (node-named e)
                rel  specific-example-of
                arg2 (node-named z))
         (build arg1 (node-named y)
                rel  (build lex know)
                arg2 (node-named k))
         (build arg1 (node-named z)
                rel  in-domain
                arg2 (node-named k)))
  cq     (build plan (build agent            (node-named x)
                            verb             ask-about
                            indirect-object  (node-named y)
                            object           (node-named si))
                goal (build arg1 (node-named x)
                            rel  (lex want)
                            arg2 (build arg1 (node-named x)
                                        rel  (build lex know)
                                        arg2 (node-named k)))))
```

Figure 2.13.5: Rule 1 from the Knowledge Base
If x asks y about *si*, which is a situation-index to the situation e, a specific example of z (and there's a node k, which is y's knowledge of z), then the asking-about was a plan for that goal.

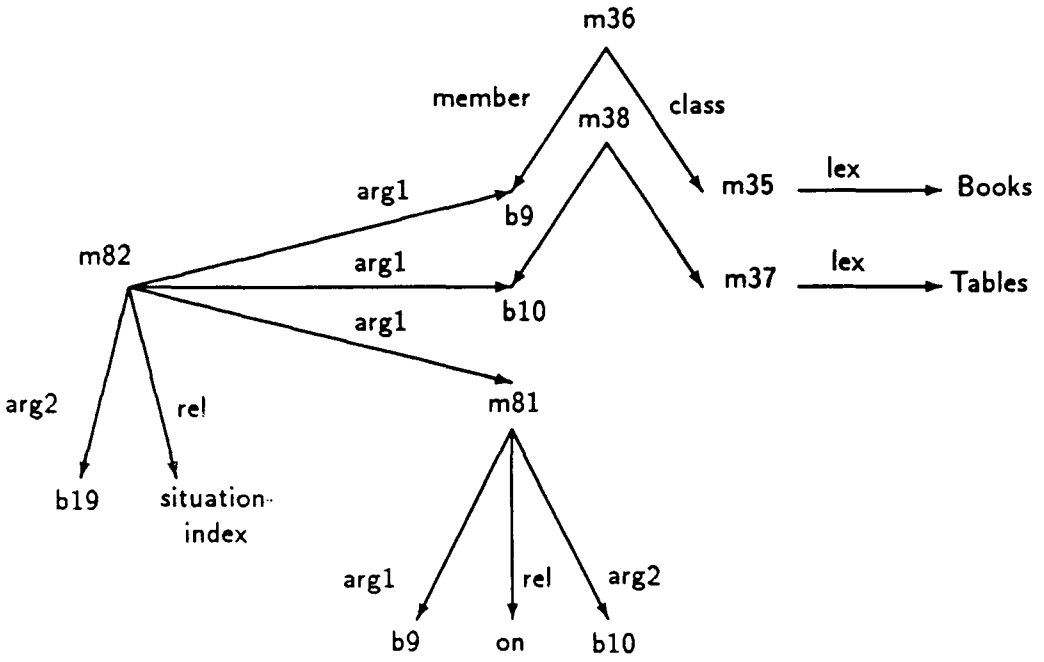

Figure 2.13.6: The representation of "e," the example referred to in the first question with the phrase "the book on the table?"

The next two antecedents of rule 1 involve a variable "k" which can be thought of as the pupil's ("y's") knowledge of statics. The variable "k" gets bound to a node built by another rule, part of the program's general world knowledge, that says there always exists a node to represent someone's knowledge of something. The backward chaining through rule 1 is now complete. The consequent is asserted, which lets the system state that Nancy wanted to know Tom's knowledge of statics.

This section has gone through part of one inference, the one which answers the first question, in order to demonstrate the general technique, and to show the representation of tutoring knowledge and Physics knowledge in the SNePS formalism. The complete knowledge base is, naturally, much larger, and therefore is not included in this report.

```
(assert forall ((new-variable-node s)
                (new-variable-node x)
                (new-variable-node y)
                (new-variable-node f))
        &ant    ((build arg1 (node-named s)
                       rel  situation)
                 (build arg1 (node-named s)
                       rel  situation-element
                       arg2 ((node-named x)
                             (node-named y)))
                 (build member (node-named x)
                       isa    Physical-Objects)
                 (build member (node-named y)
                       isa    Physical-Objects)
                 (build member (node-named f)
                       isa    Forces)
                 (build rel  exerts
                       arg1 (node-named x)
                       arg2 (node-named y)
                       arg3 (node-named f))
                 (build rel  exerts
                       arg1 (node-named y)
                       arg2 (node-named x)
                       arg3 (node-named f)))
        cq      (build arg1 (node-named s)
                      rel  specific-example-of
                      arg2 Statics))
```

Figure 2.13.7: Rule 4 from the Knowledge Base
If s is a situation, and x & y are elements of s and are Physical-Objects, and f is a force, and *exerts(x,y,f)* and *exerts(y,x,f)* then s is a specific example of *Statics*.

2.14 USING SNePS AT UMASS

Any time two research groups from different institutions attempt to cooperate there will be difficulties as well as benefits. The UMass group feels that the benefits of working with the SUNY Buffalo group were primarily intellectual: we were able to discuss our approach to problems with interested, knowledgeable, and objective researchers; it was particularly refreshing to work with a group that comes from a different background and therefore challenged many of our assumptions about possible solutions. In retrospect, both groups feel that we should have met more often so as to derive greater benefit from this interaction.

While we were all pleased with the intellectual interaction, there were difficulties with system sharing, principally because the initial port of SNePS-2 was of a version too early in its development and testing. Bugs were found, tracked down and fixed by both groups simultaneously. This became especially difficult when it became apparent that the two groups had such differences in philosophies of programming style that they refused to use each other's changes. These differences involved: how extensively macros should be used; whether Common Lisp functions like `PUSH` and `POP` should be shadowed in user-level packages; whether Common Lisp characters like `#` and `*` should be used for other purposes in the SNePS User Language.

2.15 DISCUSSION AND FUTURE WORK

Our goal is to model a rational cognitive agent whose behavior is driven by its beliefs, desires, and intentions. We want our agent to understand natural language, reason about beliefs, act rationally based on its beliefs, recognize plans, and do plan based text generation. Doing all these tasks in a single coherent framework poses several constraints. We are discovering that SNePS and its underlying theories contribute effectively towards our goal. We have designed and implemented intensional propositional representations for plans. This is a major advance over operator-based descriptions of plans. Operator-based formulations of actions tend to alienate the discussion of operators themselves. Operators are usually specified in a different language than that used for representing beliefs about states. Moreover, plans (or procedural networks) constructed from these operators can only be accessed by specialized programs (critics, executors) and, like operators, are represented in still another formalism. Our representations for acts, actions, goals, and plans build upon and add to the intensional propositional representations of SNePS. This framework enables us to tackle various tasks mentioned above in a uniform and coherent fashion.

Our current system is being advanced in several directions. In the context of planning, there are issues associated with conjunctive goals [34], non-linear plans [23, 33, 4], and dealing with the effects of actions.

Language used in planning contexts, is slightly more constrained than in arbitrary discourse. Sentences describing plans tend to be declarative, with a syntactically decomposable structure involving goal, effect, and plan definition. Handling reference is simplified by the assumption that common noun phrases correspond to typed variables. Indefinite noun phrases introduce new variables, definite noun phrases refer to previously introduced variables. Natural language generation of plans and rules involves careful selection of relevant attributes of these variables.

Sensory Acts and External Events

So far we have concentrated on the problem of designing representations suitable for discussing, using, and recognizing plans. We have demonstrated their use in a single-agent world. We are now ready to explore issues involved in using our representations to model rational cognitive agents that are capable of acting in the real world. The real world is constantly undergoing change in the presence of several agents (including the modeled one) as well as by natural phenomena. In order to behave as rational agents in a real world they should be endowed with appropriate sensors as well as effectors. Thus we are ready to explore issues concerning sensory acts, external events, and how an agent's beliefs, desires, intentions, and actions are affected by them.

Integrating Inference and Acting

In our current model (and in other state-of-the-art systems) reasoning is preformed by some inference engine and acting is done under the control of an acting executive. In order to achieve our goals we have come to the conclusion that inference and acting need to be more tightly coupled. A survey of most systems will reveal that it is somewhat awkward to do acting in reasoning (or logic-based) systems (but it is convenient to talk about representational and reasoning issues), and it is awkward to research reasoning and representational issues in systems designed for acting/planning. We are beginning to take the viewpoint that logical reasoning rules implicitly specify the act of believing, and the process of reasoning can be treated as specialized (more efficient) acting. This will enable us to integrate the acting and inference engines that can be driven by regular reasoning rules as well as connectives that will transduce a belief status to an intention-to-act status. We are currently designing such connectives. Thus our future research will attempt to clarify the relationship between inference and acting. This integrated approach used in conjunction with the principles underlying propositional semantic networks will preserve the power of acting, as well as reasoning systems, and provide a richer framework within which one can experiment with various modeling issues in AI.

Structured Variables

Another direction for our future research involves a reexamination of the representation of variables in SNePS. Consider again the conditional plan,

$$\forall x, y[Block(x) \wedge Support(y) \wedge On(x, y) \Rightarrow$$

$$GoalPlan(Clear(y), Sequence(Pickup(x), Put(x, Table)))].$$

As in the FOPC representation of this rule, the SNePS representation contains the subexpression $Pickup(x)$. Although in SNePS, the variable x is connected in the network to its restriction, $Block(x)$, the variable x is still an "atomic" node, and the term $Pickup(x)$ does not contain the restriction on x as a subterm of it. The significance of this is that the act "pick up a block" is not represented by a single term in the plan expression. Compare this representation to something like:

GoalPlan(Clear(y:Support),
Sequence(Pickup(x:Block s.t. On(x, y:Support)),
Put(x:Block s.t. On(x, y:Support), Table))).

Here, each sub-expression is conceptually complete. For example, *Pickup(x:Block s.t. On(x, y:Support))* clearly represents the act of picking up a block that is on a support. (It should be noted that the SNePS representation, using a network syntax, would not be as redundant as the linear representation.) We plan to investigate these representational issues further.

Applications to Simulated Agent

A more application-oriented direction we may pursue is to apply our techniques of representing and reasoning about plans to simulate some human agent, and to try to predict what that human agent would do in certain hypothetical circumstances.

Increased Plan Recognition

In order to recognize conditional or iterative plans, the relevant conditions which are true at the time the reported acts were performed must also be reported in addition to the reported acts. The deduction rules for plan recognition would have to be extended to include the control actions snif and sniterate.

Since SNACTor's acting executive automatically schedules acts to achieve preconditions of other scheduled acts, presumably other agents do the same thing. *I.e.,* a plan recognizer should use the precondition rules to account for those acts which are not explicit in the plan-act or plan-goal rules.

The belief revision system SNeBR can be used to implement incremental plan recognition. The belief revision system can be used to discriminate between potential plans or goals by establishing a separate context for each plan (or combination of plans) and keeping track of the consistency of each context as more and more acts about the agent are reported. The number of assumptions supporting a conclusion that the agent is performing a particular plan or pursuing a particular goal indicates the probability of that conclusion and thus can also be used to discriminate among competing plans or goals.

Understanding More Complex Paragraphs

We have shown that Model Anaphora requires more complex representation and more subtle inference than other sorts of anaphora. Anaphora that involve human interaction, plans, and goals are more complex and subtle yet. The paragraph below is typical of such text.

> (1) Last week, 32 shabby peasants appeared at the gates of the U.S. embassy in Moscow. (2) They put a startling request to American diplomats: help us get out of Russia. (3) Local authorities in Siberia had threatened to imprison the adult members of the peasants' religious group. (4) With the vague notion that a foreign embassy might help them, the Siberians had come by train to Moscow. (5) The Americans listened sympathetically, but Ambassador Foy Kohler had to stick to regulations. (6) He called the Soviet Foreign Ministry, explained the situation and asked that the peasants be removed. (7) Embarrassed by the whole thing, the U.S. officials prevented foreign correspondents from photographing or speaking with the visitors.

Two distinct Model Anaphora are used in this paragraph: "the situation in sentence (6) and "the whole thing" in sentence (7). Notice that "the whole thing" is more inclusive

than "the situation." Since our algorithm resolves each Model Anaphor in the course of processing the sentence in which it appears, the additional structure that is part of "the whole thing" will be built after "the situation" has been resolved, thus capturing this difference.

As described in Section 2.12, we find the situations by first finding the situation index, which is generally the topic of the paragraph. In the tutoring paragraph, the topic was determined to be what was in focus by the second sentence; this happened to be the indirect question of the first sentence, namely, "if an inanimate object exerts a force." However, the topic of a paragraph need not always be given in the first sentence, so we need a more sensitive mechanism for recognizing it than checking the focus at a particular moment.

The topic is generally strongly marked by some action. This may be, for example, a demonstative action, such as pointing, or, as in the tutoring paragraph, a speech act (Nancy asking a question). In the current paragraph, the topic is also marked by a speech act:

> (1) Last week, 32 shabby peasants appeared at the gates of the U.S. embassy in Moscow. (2) They *put a* startling *request* to American diplomats: *help us get out of Russia.*

To put a request is to perform a speech act; the object of the request is a likely candidate for the topic of the paragraph and is selected as the topic because it is the earliest candidate in the text.

This analysis shows that the topic is functionally, rather than just structurally, determined. To get a better grasp on how "topic" is defined and how the topic of a paragraph is determined, as well as related issues in resolving Model Anaphora, we suggest that future research include testing modifications to the structure of the paragraph to discover how different rhetorical or syntactic structures result in different specifications for the Model Anaphora.

In Figure 2.15.1 we present our preliminary research into the goals and plans that may be present in this paragraph. Future research should include refinement and implementation of this work.

Integrating Systems

The work at UMass was not as well integrated with the work at UB as we would have liked. However, merging the two streams of research offers some exciting possibilities for the future. In particular, the UB work on describing plans in natural language would have many applications in a system designed to read and understand paragraphs. In the tutoring domain, we could describe to the system condition-action pairs corresponding to tutoring strategies. These condition-action pairs would later help the system recognize these strategies when they are employed in the tutoring situations described in paragraphs.

(1) None.

(2)	Peasants	*Goal*	be out of Russia
		Plan	ask American authorities for help
(3)	Authorities	*Goal*	preventing peasants' religious practices
		Plan	threaten peasants
	Peasants	*Goal*	keeping religion
		Plan	??
(4)	Peasants	*Subgoal*	ask a foreign embassy for help
		Plan	go to embassy in Moscow

(This is an elaboration of the plan in (2).)

(5, 6)	Americans	*Goal*	stick to regulations
		Plan	ask Russians to remove peasants
		Goal	be nice to peasants
		Plan	act sympathetic
(7)	Americans	*Goal*	cover up embarrassment
		Plan	keep reporters away

Figure 2.15.1: Plans and Goals in the "Shabby Peasants" paragraph

2.16 TRIPS FUNDED BY RADC

Workshop on Formal Foundations of Semantic Networks, Santa Catalina Island, CA, February 15-18, 1989: Shapiro

RADC Sponsor Meeting with U. Mass., Rome, NY, February 23, 1989: Shapiro, Kumar, Ali, Haas

NAIC Committee Meeting, Rochester Institute of Technology, April 14, 1989: Shapiro

First International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Ontario, Canada, May 15-18, 1989: Shapiro

NAIC Committee Meeting, Syracuse, NY, June 28, 1989: Shapiro

RADC briefing and demonstration with U.Mass, Rome, NY, July 27, 1989: Shapiro, Kumar Ali

Eleventh International Joint Conference on Artificial Intelligence (IJCAI), Detroit, Michigan, August 20-25, 1989: Shapiro

Annual NAIC Meeting, Minnowbrook, NY, August 14-17, 1989: Shapiro

2.17 NLP PUBLICATIONS IN 1989

Shapiro, S. C. and The SNePS Implementation Group, "SNePS-2 User's Manual", Department of Computer Science, SUNY at Buffalo.

Shapiro, S. C., (unpublished) "Formal Foundations of an Intensional Propositional Semantic Network", Presented at the Workshop on Formal Aspects of Semantic Networks, Santa Catalina Island, CA.

Shapiro, S. C., Kumar, D. and Ali, S., "A Propositional Network Approach to Plans and Plan Recognition", *Proceedings of the 1988 Workshop on Plan Recognition*, AAAI-88, Morgan Kaufmann, Los Altos, CA.

Shapiro, S. C., "The CASSIE Projects: An Approach to Natural Language Competence", to appear in *Proceedings of the 4th Portugese Conference on Artificial Intelligence*, to be held in Lisbon, Portagul, September 36–30, 1989.

Anderson, S. D., "Speaking of Elephants: Generating from Knowledge Representations", COINS Technical Report 89-19, Computer and Information Science Department, University of Massachusetts, Amherst, MA 1989.

Woolf, B. P., "Hypermedia in Education and Training", *Proceedings of the Artificial Intelligence and Intelligent Tutoring Systems Symposium*, University of Maine, Orono, ME, 1989.

Woolf, B. P., "Representing, Acquiring and Reasoning about Tutoring Knowledge," *Proceedings of the Air Force Workshop on Intelligent Tutoring Systems*, Southwestern Research Institute, San Antonio, TX 1989.

Sibun, P. and Huettner, A. K. "Spatial Deixis in Generating Description," COINS Technical Report 89-19, Computer and Information Science Department, University of Massachusetts, Amherst, MA 1989.

2.18 SUNY at BUFFALO STUDENTS DIRECTLY FUNDED BY NAIC, 1985–1989

(Names in **bold** are supported in 1989 by RADC at SUNY Buffalo)

James Geller, Ph.D.	2/85 - 5/88	Research Assistant
Academia, USA	6/88 - 7/88	Research Associate
Permanent Resident (applied after graduation)		
Mingruey Taie, Ph.D.	2/85 - 5/87	Research Assistant
Industry, USA	5/87 - 8/87	Research Associate
Permanent Resident (applied after graduation)		
Joao P. Martins, Ph.D.	5/85 - 7/85	Research Associate
Academia, Portugal	Post Doc. work	Visiting Asst. Professor
Foreign		
Scott S. Campbell, MS	8/85 - 9/87	Research Assistant
American	9/87 - 8/89	Programmer/Analyst
Jiah-shing Chen, MS	7/87 - 8/89	Research Assistant
Foreign		
Amruth Kumar N.	1/88 - 8/89	Research Assistant
Foreign		
Joongmin Choi	8/88 - 8/89	Research Assistant
Foreign		
Deepak Kumar, MS	8/87 - 8/89	Research Assistant
Foreign		
Syed S. Ali, MS	8/87 - 8/89	Research Assistant
Canadian		
Juergen Haas	1/89 - 8/89	Research Assistant
Foreign		
Sudip Nag	8/88 - 7/89	Research Assistant
Permanent Resident		

Byung S. Yoo, MS Permanent Resident Industry, IBM	7/88 - 10/88	Research Assistant
David B. Satnik American Industry, Seattle, WA	7/87 - 5/88	Research Assistant
Richard W. Wyatt Permanent Resident Ph.D., Psychology Student, SUNY at Buffalo	8/87 - 1/88	Research Assistant
Keith E. Bettinger, MS American Student, SUNY at Buffalo	5/87 - 8/87	Research Assistant
Kwong Yiu Yim Foreign	1/86 - 5/86	Research Assistant
Jennifer M. Suchin, MS American Industry, Pittsburgh, PA	6/85 - 8/85	Project Aide (Graduate)

RADC Supported Ph.D.'s in Progress, SUNY at Buffalo

Kumar, Deepak
Planning in SNePS

Chen, Jiah-shing
Model-based Diagnosis Using Multiple Approaches

Choi, Joongmin
An Intelligent Reasoning System by Knowledge Migration and Shadowing

Kumar N., Amruth
Issues in Diagnosis: Sequential and Combinational

2.19 DEPARTMENT STATISTICS, SUNY at BUFFALO: ARTIFICIAL INTELLIGENCE

YEAR	AI Ph.D.'s	Non-AI Ph.D.'s	AI Master's	Non-AI Master's
1989 (through June)	None	None	21	11
1988	5	None	23	12
1987	3	None	28	16
1986	1	2	24	7
1985	2	1	28	3
1984	None	2	27	9
1983	2	1		

2.20 PH.D. GRADUATES IN ARTIFICIAL INTELLIGENCE, SUNY at BUFFALO

(Names in **bold** were supported by RADC)

1988

James Geller	Assistant Professor	New Jersey Institute of Technology
Jonathan J. Hull	Research Assistant Professor	SUNY at Buffalo Buffalo, NY
Ganapathy Krishnan	Assistant Professor	Stetson University Deland, FL
Ching-Huei Wang	Research Analyst	Boeing Electronics Corp. Seattle, WA
Zhigang Xiang	Assistant Professor	Dept. of Computer Science Queen's College (CUNY) New York, NY

1987

Michael J. Almeida	Assistant Professor	Penn State University Dept. of Computer Science Whitmore Lab University Park, PA
George Sicherman		AT&T Bell Labs Middletown, NJ
Mingruey R. Taie		AT&T Bell Labs Middletown, NJ

1986

Ernesto Morgado	Assistant Professor	Dept. de Engenaria Mechanica Instituto Superior Tenico Lisbon, Portugal

1985

Radmilo M. Bozinovic		GO Corporation San Francisco, CA
Jeannette Neal	Research Assistant Professor	Calspan Corporation Cheektowaga, NY Research (CUBRC)

2.21 MASTER'S DEGREES FROM THE DEPARTMENT OF COMPUTER SCIENCE, SUNY at BUFFALO (1984–1989)

(Names in **bold** were supported by RADC)

Artificial Intelligence (1989, through June)

Ansley, William	Dobes, Zuzana	Majkowski, Bruce
Banerjee, Sarbani	Goldberg, Neal	**Nag, Sudip**
Chaluplsky, Hans	Gucwa, John R.	Po, Cherng-Fong
Chang, Han Yi	Jain, Naresh Kumar	Sin,Via Tong
Cohen, Edward	Lee, Chieng	Tan, Wan
Colucci, Paul	Lin, Yi Chang	Wu, Li Shin
Crovella, Mark E.	Lombardo, Karen	Yen, Shyh-Guang

CS Other than AI (1989, through June)

Arora, Rajendra	Grupka, Laurette	Sherman, Paul J.
Chang, Tien	Kingsbury, Linda A.	Williams, Francine
Delgado, Diane M.	Liu, Paul C-W	Wu, W-C Jevons
Fenrich, Richard	Menon, Rajeev	

Artificial Intelligence (1988)

Bansal, Surendra	Haller, Susan	Soh, Jung
Benz, David	Hardy, Michael J.	Strohmeier, Nancy
Bettinger, Keith	Hou, Chien-Long	Swerdloff, Lucien
Biernat, Catherine	Kornacki, Edmund	Vecellio, Gary
Chang, Adam Chih-Yen	Kuan, Chic Chau	Wan, Tzu-Horng Tom
Danko, Paul Jr.	**Kumar, Deepak**	Williams, Francine
Debbins, Catherine	Lew, Kurk	**Yoo, Byung**
Govindaraju, Venugopal	Lo, Ka-Chiu	

CS Other than AI (1988)

Arora, Rajenda	Duh, Ying Ying	Lam, William
Azar, Chawki	Hosangadi, Shrikant	Mantharam, Mythili
Bahl, Vikram	Lagona, Scott	Sarraf, Elias
Desirazu, Shyam	Lakshman, T.K.	Wahl, Norman

Artificial Intelligence (1987)

Binkerhoff, Linda
Chan, Chung Man
Campbell, Scott
Chang, Cheng-Ping
Chen, Yung-Yuan
Chun, Soon Ae
DeVinney, George
Dodson-Simmons, Onda
Ehrlich, Karen
Feuerstien, Steven
Gupta, Rakesh
Jain, Hwejdar
Kim, Joong-Won
Kuo, Chung-Kuo
Lang, Su-Jin
Lee, Hui-Chung
Li, Niacong
Li, Peter
Murty, Kurella
Schneck, Nelson
Schwartz, Margaret
Siracusa, Thomas
So, Hon-Man
Thomas, Timothy
Wang, Gretchen
Wroblewski, Susan
Wu, Teng Yien
Wu, Wei-Jye

CS Other than AI (1987)

Boxer, Lawrence
Chang, Cheng-Ping
Cheng, Tony H-Y
Chow, Lawrence
Gaur, Yogesh
Girod, Allison
Gunning, Mark
Hiroi, Toshiyuki
Jang, Yong Ho
Lively, Richard
Mackey, Niloufer
Miller, Susan
Nimmagadda, Venkata
Rajan, Dayanand
Shende, Anil
Subrahmanyam, Pratap

Artificial Intelligence (1986)

Bross, Neal
Deutschlander, Kenneth
Hull, Richard
Jayanthi, Sarma
Kailar, Sudah
Krishnaswamy, Latha
Krishnaswamy, Vijaykumar
Lee, Gin-Wha
Lu, Wuhsiung
Ma, Pu-Kao
MacFadden, Douglas
McConnell, Jeffery
Murphey-Shelton, Anne
Murray, Deborah
Rastogi, Ajay
Sauciunac, Christine
Shin, Kwang Un
Shyong, Beth M-F
Swaminathan, Puducode
Ting, Hungtau
Wang, Fen-Cheng
Winkowski, Danie
Wood, Gabriel
Yang, Ching-Yun

CS Other than AI (1986)

Bharadhwaj, Rajeev
Kim, Dongsoo
Martin, Dennis
Ramshankar, J.V.
Rosenblum, Leonard
Schwartz, Mary
Vassallo, Mario

Artificial Intelligence (1985)

Allen, Kristen
Arora, Kulbir
Barback, Joseph
Baxter, William
Chen, Li-Wha
Clark, Michael
Hise, Denise
Hu, Hai Hsu
I, Chih-Li
Isaac, Reeba M.
Kalnitz, Paul
Kramarczyk (Kramer), Chrisopher
Kuo, Chi-Kai
Li, Kuang Chieh
Lo, Mie-Ying
Lung, Hsi-Hao Howard
Min, Byoung Ho
Niyogi, Debashish
Palumbo, Paul
Pawlicki, Thadeus
Saks, Victor
Suchin, Jennifer
Taie, Shwu-Fan
Wang, Ching-Ying
Wang, Der-Yuk
Wiebe, Janyce
Yang, Jin-Tan David
Yi, Myungzoon

CS Other than AI (1985)

Chi, Henjin
Fu, Jing-sheng
Olin, David

Artificial Intelligence (1984)

Chang, Chung
Chen, Kwei-Jen
Choy, Chi Chung
Das, Mangobinda
Haefner, Michael
He, Hung Chyi
Hsu, An-Mei
Jou, Chen-Jye
Kellick, Diane
Konakanchi, Krishna
Kung, Peter F.
Leu, Fang Hsiung
Lin, Han-Hong
Liu, Ming
Liu, Peter (Sai-Ming)
Lo, Yu Li
Lung, Hsi-Hong
Milich, Gregory
Nemirov, Hinda
Phillips, Gretchen
Rapaport, William J.
Shlossman, Paul
Shen, Chien-Chih
Su, Suyuan C.
Yang, Lien-Jang
Yao, Jo-Lan
Zayan, Ahme

CS Other than AI (1984)

Alsam, Javaid
He, Hung Chyi
Hung, Hing Kai
Izard, Thomas
Klee, Karl
Leung, Chun Wah
Oviedo, Enrique
Welte, Martha
Zachopoulos, George

2.22 ARTIFICIAL INTELLIGENCE FACULTY, SUNY at BUFFALO

Stuart C. Shapiro	Professor	Knowledge Representation Reasoning NL Processing
Sargur N. Srihari	Professor	Knowledge-Based Systems Computer Vision Pattern Recognition
William J. Rapaport	Associate Professor	Knowledge Representation Philosophical Foundations NL Processing
Shoshana Hardt	Assistant Professor	Expert Systems Qualitative Reasoning
David Sher	Assistant Professor	Computer Vision
Deborah K. W. Walters	Assistant Professor	Computer Vision
Richard Wildes	Assistant Professor	Computer Vision
Jonathan J. Hull	Research Assistant Professor	Computer Vision
Jeannette G. Neal	Research Assistant Professor Senior Scientist, Calspan	Intelligent Interfaces NL Understanding Expert Systems

2.23 ARTIFICIAL INTELLIGENCE ADDITIONS TC THE DEPARTMENT DURING THE PERIOD OF NAIC FUNDING, SUNY at BUFFALO

New Artificial Intelligence Faculty Appointed

Michael Leyton Assistant Professor	Computer Vision	8/86 - 8/87
David Sher Assistant Professor	Computer Vision	8/87 - present
Richard Wildes Assistant Professor	Computer Vision	9/88 - present
Jeannette G. Neal Research Assistant Professor Senior Scientist, Calspan	Intelligent Interfaces NL Understanding Expert Systems	2/88 - present
Jonathan J. Hull, Research Assistant Professor	Computer Vision	9/87 - present

New Artificial Intelligence Courses, SUNY at Buffalo

CS 514 Vision
CS 666 Introduction to Image Analysis
CS 676 Knowledge Representation

2.24 ONGOING ARTIFICIAL INTELLIGENCE DISSERTATIONS, SUNY at BUFFALO

AI Students Past the Ph.D. Primary Area Examination
(Name in **bold** is currently supported by RADC)

Arora, Kulbir
Qualitative Reasoning about Physical Systems

Baxter, William
Multiresolution Edge Detection

Ehrlich, Karen
Automatic Acquisition of Natural Language

Kumar, Deepak
Planning in SNePS

Lively, Richard
Texture Segmentation of Images

Niyogi, Debashish
A Knowledge-Bases Approach to Analyzing Logical Document Structure

Pawlicki, Thaddeus F.
A Neural Network Approach to the Indexing Problems on Model-Based Computer Vision Systems

Srihari, Rohini
Integration of Information from Visual & Linquistic Sources

Wiebe, Janyce M.
A Computational Theory of Perspective in Narrative

Yuhan, Albert Hanyong
Dynamic Computation of Reference Frames in Spatial Information Processing

Biblicgraphy

[1] A. Akmajian and F. W. Heny. *An Introduction to the Principles of Transformational Syntax.* The MIT Press, 1975.

[2] . M. J. Almeida. *Reasoning About the Temporal Structure of Narratives.* PhD thesis, Technical Report 87–10, Department of Computer Science, SUNY at Buffalo, 1987.

[3] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[4] M. Drummond and A. Tate. AI planning: A tutorial and review. Technical Report AIAI-TR-30, Artificial Intelligence Applications Institute, University of Edinburgh, Edinburgh, November 1987.

[5] M. E. Drummond. A representation of action and belief for automatic planning systems. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 189–212, Los Altos, CA, 1987. AAAI and CSLI, Morgan Kauffmann.

[6] M. P. Georgeff. Planning. In *Annual Reviews of Computer Science Volume 2*, pages 359–400. Annual Reviews Inc., Palo Alto, CA, 1987.

[7] M. P. Georgeff. An embedded reasoning and planning system. In J. Webber, J. Tenenberg, and J. Allen, editors, *Advance Proceedings of The Rochester Planning Workshop– From Formal Systems to Practical Systems*, pages 79–101, October 1988.

[8] J. Hankamer and I. Sag. Deep and surface anaphora. *Linguistic Inquiry*, 7(1):391–426, 1976.

[9] E. Hinrichs. Temporal anaphora in discourses of English. *Linguistics and Philosophy*, 9(1):63–82, 1986.

[10] A. Huettner. Unpublished Ph.D. thesis, 1989. Linguistics Department, University of Massachusetts at Amherst. A complete citation was unavailable at time of publication.

[11] K. Huff and V. Lesser. The grapple plan formalism. COINS Technical Report 87–08, University of Massachusetts, 1987.

[12] D. Kumar, S. Ali, and S. C. Shapiro. Discussing, using and recognizing plans in SNePS preliminary report - SNACTor: An acting system. In *Proceedings of the Seventh Biennial Convention of South East Asia Regional Confederation*, pages 177–182, New Delhi, India, 1988. Tata McGraw-Hill.

[13] A. L. Lansky. A representation of parallel activity based on events, structure, and causality. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 123–160, Los Altos, CA, 1987. AAAI and CSLI, Morgan Kauffmann.

[14] V. Lifschitz. On the semantics of STRIPS. In M. P. Georgeff and A. L. Lansky, editors, *Reasoning about Actions and Plans - Proceedings of the 1986 Workshop*, pages 1–10, Los Altos, CA, 1987. AAAI and CSLI, Morgan Kauffmann.

[15] J. P. Martins and S. C. Shapiro. Hypothetical reasoning. In *Applications of Artificial Intelligence to Engineering Problems: Proceedings of The 1st International Conference*, pages 1029–1042, Berlin, 1986. Springer–Verlag.

[16] J. P. Martins and S. C. Shapiro. A model for belief revision. *Artificial Intelligence*, 35(1):25–79, 1988.

[17] D. P. McKay and S. C. Shapiro. Using active connection graphs for reasoning with recursive rules. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 368–374, Los Altos, CA, 1981. Morgan Kaufmann.

[18] E. J. Morgado and S. C. Shapiro. Believing and acting: A study of meta-knowledge and meta-reasoning. In *Proceedings of EPIA-85 ("Encontro Portugues de Inteligencia Artificial")*, pages 138–154, Oporto, Portugal, 1985.

[19] N. J. Nilsson. *Principles Of Artificial Intelligence.* Tioga Publishing Company, Palo Alto, CA, 1980.

[20] A. Paivio, J. Yuille, and S. Madigan. Concreteness, imagery, and meaningfulness values for 925 nouns. *Journal of Experimental Psychology Monograph Supplement*, 76:1–25, 1968.

[21] B. Partee. Nominal and temporal anaphora. *In Linguistics and Philosophy*, 7(3):243–286, 1984.

[22] E. Rosch, C. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *In Cognitive Psychology*, 8:382–439, 1976.

[23] E. D. Sacerdoti. *A Structure for Plans and Behavior.* Elsevier North Holland, New York, NY, 1977.

[24] S. C. Shapiro. The SNePS semantic network processing system. In N. V. Findler, editor, *Associative Networks: The Representation and Use of Knowledge by Computers*, pages 179–203. Academic Press, New York, 1979.

[25] S. C. Shapiro. Generalized augmented transition network grammars for generation from semantic networks. *The American Journal of Computational Linguistics*, 8(1):12–25, 1982.

[26] S. C. Shapiro. Symmetric relations, intensional individuals, and variable binding. *Proceedings of the IEEE*, 74(10):1354–1363, 1986.

[27] S. C. Shapiro. Representing plans and acts. In *Proceedings of the Third Annual Workshop on Conceptual Graphs*, pages 3.2.7–1 – 3.2.7–6. The American Association for Artificial Intelligence, Menlo Park, CA, 1988.

[28] S. C. Shapiro, D. Kumar, and S. Ali. A propositional network approach to plans and plan recognition. In *Proceedings of the 1988 Workshop on Plan Recognition*, page 21, Los Altos, CA, 1989. Morgan Kaufmann.

[29] S. C. Shapiro, J. Martins, and D. McKay. Bi-directional inference. In *Proceedings of the Fourth Annual Meeting of the Cognitive Science Society*, pages 90–93, Ann Arbor, MI, 1982.

[30] S. C. Shapiro and W. J. Rapaport. SNePS considered as a fully intensional propositional semantic network. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier*, pages 263–315. Springer–Verlag, New York, 1987.

[31] S. C. Shapiro, B. Woolf, D. Kumar, S. S. Ali, P. Sibun, D. Forster, and S. Anderson. Discussing, using, and recognizing plans–Annual Report for 1988. Technical report, North-East Artificial Intelligence Consortium, 1989.

[32] C. Sidner. Towards a computational theory of definite anaphora comprehension in English discourse. Technical Report TR-537, Massachusetts Institute of Technology, 1979.

[33] A. Tate. Generating project networks. In *Proceedings 5th IJCAI*, pages 888–93, 1977.

[34] R. Waldinger. Achieving several goals simultaneously. In D. Michie, editor, *Machine Intelligence 8*, pages 94–136. Ellis Horwood, Chichester, England, 1977.

[35] B. Webber. So what do we talk about now? In M. Brady and R. Berwick, editors, *Computational Models of Discourse*, pages 331–371. The MIT Press, 1983.

[36] B. Webber. Event reference. In *Position Papers for TINLAP-3: Theoretical Issues in Natural Language Processing-3*, pages 137–142, Las Cruces, NM, 1987.

[37] B. Webber. Discourse anaphora. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 113–122, Buffalo, NY, June 1988. ACL.

[38] D. E. Wilkins. *Practical Planning–Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, Palo Alto, CA, 1988.

Appendix 2.A TOM Grammar

2.A.1 Verbal Elements in the SNePS ATN

Our first attempt at handling verbal elements in English was a failure. We were trying hard to go forward in a principled way, from CFG[1] + TG[2] rules to either a pure CFG or an FSM.[3] The problems mostly arose because as we multiply out the CFG + TG rules, we first get a hefty explosion of rules, which then must be boiled down by looking for identities or similarities between rules or states. Similarities between rules or states don't really result in any reduction in the number of rules or states, but there is a reduction in the size of the NOTATION, because you can parameterize the rules or states, resulting in an identity, which can then be collapsed. But this is still really hard, and properly collapsing the notation was very confusing.

This grammar presents a new approach, by trying to go directly from the CFG + TG rules to an ATN, which is in consonance with the original intention of Woods' ATN formalism. In fact, we will end up parameterizing the rules even more than they are already! These rules are primarily drawn from [1].

We start with:

> S → NP VS VP-REST
> VS → (MODAL) (have) (be) V
> plus the TG rules of Aux-Hopping, Negation, and Question Formation. And don't forget DO-support.

First, we note that for each of the auxiliaries, much the same thing is happening on the resulting ATN arc, to wit: if the auxiliary is of the right category (MODAL, 'have,' or 'be'), and the auxiliary is in the right form (does it have the suffix which has hopped in from the left?), traverse the arc. So let's simplify the CFG rule and add a constraint.

> VS → AUX* V
> AUX → MODAL | have| be
> Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.'

The rule of Negation adds that if the auxiliary is the first one, it can be followed by a negative morpheme. That simply goes into the constraint:

> VS → AUX | not * V
> AUX → MODAL | have | be
> Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.' Optionally, 'not' immediately follows the first auxiliary.

[1]Context-Free Grammar
[2]Transformational Grammar
[3]Finite State Machine

Let's throw in DO-support:

VS → AUX | not * V
AUX → do | MODAL | have | be
Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.' If 'do' is an auxiliary, then it is the first and only auxiliary (if you don't believe in DO-support for positive declaratives, then 'do' must be followed by 'not'). Optionally, 'not' immediately follows the first auxiliary.

The second step is going to look weird, but bear with me. We want to handle Question Formation. So, let's eliminate the ordering of subject and auxiliary from the CFG rules and put it into the constraint:

S → VS VP-REST
VS → NP | AUX | not * V
AUX → do | MODAL | have | be
Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.' If 'do' is an auxiliary, then it is the first and only auxiliary (if you don't believe in DO-support for positive declaratives, then 'do' must be followed by 'not'). Optionally, 'not' immediately follows the first auxiliary. NP appears exactly once and precedes or follows the first auxiliary (together with 'not').

The third step is to convert this stuff into an ATN. We only need two states to deal with the VS rule (the S rule is obvious), with the second of those states being trivial (it reads the V). The arcs of the first state are: PUSH NP, CAT V, CAT NEG, and JUMP. Why the JUMP? That's because many of these items are optional, and the JUMP is like an epsilon transition in an NFA.

The constraints are implemented via registers in the ATN, plus lisp code in the tests and actions on the arcs. The information about modals and auxiliaries is available when the node is finally built (in VP-END), but is discarded at that point, since we don't know how to represent tense, aspect and modal information. If someone has an idea, the information is all there.

2.A.2 The ATN

```
(S
  ;; The words 'no' and 'yes' can be sentences on their own
  (wrd (yes no) t
       (setr propositional-anaphor |*|)
       (to YES-NO))
  (cat wh-adjunct t
       (initialize-vs-machine)
       (setr mood :why-question)
       (to VS))
  (jump VS t
        (initialize-vs-machine)))


(YES-NO
  (pop (progn (lisp::push (cons 'contains-propositional-anaphora? 't)
                          sneps::*sentence-facts*)
              (setr net
                    (build
                      equiv1 (lisp-escape
                               (new-base-node some-proposition))
                      equiv2 (lisp-escape
                               (if (string-equal
                                     "yes"
                                     (getr propositional-anaphor))
                                   (build lex true)
                                   (build lex false)))))
              (getr net))
       t))


(VS (PUSH NP (and (null (getr subj))
                  (member (length (getr auxes)) '(0 1)))
          (sendr v v)
          (setr prev (getr |*|))
          (setr subj (getr |*|))
          (new-case-frame :agent '|*|)
          (if (and (null (getr auxes))
                   (null (getr v)))
              (setr mood :declarative)
              (if (null (getr mood))
                  (setr mood :yes-no-question)))
          (TO VS))
```

```
(CAT MODAL (acceptable-verbal-element? :modal)
     (process-auxiliary :modal)
     (TO VS))
(CAT V (acceptable-verbal-element? :do)
     (process-auxiliary :do)
     (TO VS))
(CAT V (acceptable-verbal-element? :have)
     (process-auxiliary :have)
     (TO VS))
(CAT V (acceptable-verbal-element? :be)
     (process-auxiliary :be)
     (TO VS))
(CAT NEG
     (and (eql (length (getr auxes)) 1)
          (member (getf ctgy (getr prev)) '(v modal))
          ;; the following means the ATN doesn't allow double
          ;; negatives, such as 'John can't not go to the party.'
          ;; or 'John can't be not going to the party.'
          (null (getr neg)))
     (setr prev *current-word*)
     (setr neg t)
     (TO VS))
(CAT V
     (and (acceptable-verbal-element? :v)
          ;; The only main verbs that can appear before the subject
          ;; are 'have' and 'be.'  See the test suite for examples.
          (or (not (null (getr subj)))
              (member (getr |*|)
                      '("have" "be") :test #'string-equal)))
     (talk 2 "main verb")
     (setr v (get-lexical-feature 'root (getr features)))
     (when (null (getr auxes))
       (internal-setr
         neg
         (get-lexical-feature :negative (getr features))))
     (if (and (null (getr subj))
              (null (getr mood)))
         (setr mood :yes-no-question))
     (TO VS))
(JUMP VP-ARGS (and (not (null (getr subj)))
                   (not (null (getr v)))
                   (typep (getr mood) 'legal-mood))))
```

```
;;; This node collects arguments of the verb.

(vp-args
  (push np (or (getf trans v) (getf obj+pplocative v))
        (sendr v v)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (to vp-args))
  (wrd if (getf condv v)
       (to vp-if))
  (cat adj (equal (getf root (getr v)) "be")
       (setr predicate |*|)
       (setr type :predicate-adjective)
       (to vp-adjuncts))
  (push scomp (getf infinitive-scomp v)
        (sendr case-frames case-frames)
        (sendr subj obj)
        (setr iobj obj)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (new-case-frame :indirect-object 'iobj)
        (to vp-adjuncts))
  (push s (getf scomp v)
        (sendr case-frames case-frames)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (to vp-adjuncts))
  (cat prep (member (getr |*|) (getf :particles (getr v))
                    :test #'string-equal)
       (setr particle |*|)
       (to vp-particle))
  (push pp  t
        (sendr v (getr v))
        (addr vmods |*|)
        (let ((node  (cadr (getr |*|)))
              (role  (intern (the (and symbol (not null)) (car (getr |*|)))
                             *keyword-package*)))
          (new-case-frame role node))
        (to vp-args))
  (jump vp-adjuncts t))


;;; This node ought to collect adjuncts to the verb, such as time/location
```

```
;;; NPs and PPs or rationale clauses.  It only does the last, since the PPs
;;; are treated uniformly as vmods, and therefore are collected in the
;;; VP-ARGS node.
(vp-adjuncts
  (push rationale-clause t
        (sendr subj (getr subj))
        (sendr case-frames case-frames)
        (setr rationale |*|)
        (to vp-end))
  (jump vp-end t))


;;; This node assumes that if you have two NPs, the one preceding the
;;; particle is the IOBJ and the one following is the OBJ.
(vp-particle
  (push np t
        (sendr v v)
        (setr iobj obj)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (setf (car (member :object (getr case-frames))) :indirect-object)
        (to vp-args))
  ;; For sentences like ''call John up''
  (jump vp-args t))


(vp-end
  (pop (progn
         (setr net
               (lisp-escape
                 (if (eq (getr type) :predicate-adjective)
                     (eval
                       `(build ,@(if *increment-time?*
                                     `(stime   ,(node-named sneps::stm)
                                       etime   ,(node-named sneps::etm)))
                               rel
                               ,(build lex (lisp-escape (getr predicate)))
                               arg1
                               ,(lisp-escape (getr subj))))
                     (eval
                       `(build ,@(if *increment-time?*
                                     `(stime   ,(node-named sneps::stm)
                                       etime   ,(node-named sneps::etm)))
```

```
                        agent
                        (lisp-escape (getr subj))
                        verb
                        ,(if (getr particle)
                             (sneps-symbol
                               (string-append
                                 (getr v) "-" (getr particle)))
                             '(build lex (lisp-escape (getr v))))
                        ;; if an arc would point to NIL,
                        ;; it's not built
                        indirect-object (lisp-escape (getr iobj))
                        object    (lisp-escape (getr obj))
                        rationale (lisp-escape (getr rationale))
                        . ,(lisp-escape (getr vmods)))))))
(if (not (null (getr neg)))
    (setr net (lisp-escape
                (eval
                  '(build rel  (build lex not)
                              arg1 ,(getr net))))))
(setr net
      (lisp-escape
        (ecase (getr mood)
          (:declarative     (getr net))
          (:yes-no-question
           (eval '(build rel  (build lex truth-value)
                              arg1 ,(getr net))))
          (:why-question    (eval '(build rel  (build lex why)
                                               arg1 ,(getr net))))
          (:how-question    (eval '(build rel  (build lex how)
                                               arg1 ,(getr net)))))))
(new-case-frame :case-frame 'net)
(liftr case-frames case-frames)
;; This works because VP-END will be the last node executed,
;; so the case-frames register will be right.
(setq sneps::*case-frames* (getr case-frames))
(when *increment-time?*
  (build after  (new-base-node sneps::nstm)
         before (node-named    sneps::stm))
  (build after  (new-base-node sneps::netm)
         before (node-named    sneps::etm))
  (remember-node (node-named sneps::netm) sneps::etm)
  (remember-node (node-named sneps::nstm) sneps::stm))
```

```
        (getr net))
       (or (and (getf intrans v)
                (nullr obj)
                (nullr iobj))
           (and (getf trans v)
                (not (nullr obj))
                (nullr iobj))
           (and (getf scomp v)
                (not (nullr obj))
                (nullr iobj))
           (and (getf infinitive-scomp v)
                (not (nullr obj))
                (not (nullr iobj)))
           (and (getf obj+pplocative v)
                (not (nullr obj))
                (nullr iobj)
                (not (nullr vmods))
                (contains-locative-pp? (getr vmods)))
           (and (getf small-clause v)
                (not (nullr obj))
                (nullr iobj))
           (and (getf condv v)
                'what??)
           (and (getf prop-att v)
                'what??))))


(vp-if
  (push s t
        (sendr case-frames case-frames)
        (setr type 'tv)
        (setr iobj obj)
        (setr obj  (lisp-escape (build arg1 (lisp-escape (getr |*|))
                                       rel  (build lex truth-value))))
        (new-case-frame :if-obj 'obj)
        (sneps::= (lisp-escape (find arg2- (lisp-escape (getr obj))))
                  some-proposition)
        (to vp-args)))


(scomp
  (jump scomp-subj t))
```

```
(scomp-subj
  (wrd to t
       ;; This is much like (initialize-vs-machine), but omits some things
       ;; because we're in a more constrained situation.
       (setr neg  nil)
       (setr mood nil)
       (setr v nil)
       (to scomp-to)))


(scomp-to
  ;; the following means the ATN doesn't allow double negatives in
  ;; infinitive complements,
  ;; such as 'Mary requested John to not not go to the party.'
  (CAT NEG (null (getr neg))
       (setr neg t)
       (TO SCOMP-TO))
  (CAT V ([u]member :infinitive
                   (get-lexical-feature :untensed-forms (getr features))
                   englex:*legal-untensed-forms*)
       (setr v |*|)
       (setr mood :declarative)
       (to scomp-v)))


(scomp-v
  (push vp-args t
        (sendr subj subj)
        (sendr v    v)
        (sendr neg  neg)
        (sendr mood mood)
        (sendr case-frames case-frames)
        (setr  predicate |*|)
        (to scomp-end)))


(scomp-end
  (pop (progn (liftr case-frames)
              (getr predicate))
       t))


(comp
  (cat v ([u]member :infinitive
```

```
                    (get-lexical-feature :untensed-forms (getr features))
                    englex:*legal-untensed-forms*)
       (setr v |*|)
       (to vp-args)))


(rationale-clause
  (wrd to t
       (to rationale-clause-v)))


(rationale-clause-v
  (cat v ([u]member :infinitive
                    (get-lexical-feature :untensed-forms (getr features))
                    englex:*legal-untensed-forms*)
       (setr v |*|)
       (to rationale-clause-args)))


;; doesn't deal with double-object verbs:
(rationale-clause-args
  (push np (and (getf trans v)
                (not (getr obj)))
        (sendr v v)
        (setr obj |*|)
        (new-case-frame :rat-obj 'obj)
        (to rationale-clause-adjective))
  (jump rationale-clause-adjective t))


(rationale-clause-adjective
  (cat adj (getf small-clause v)
       (progn "to deal with elided small-clause subjects, as in
              'john lifted weights to keep <John> strong'"
              (if (null (getr obj))
                  (setr obj subj)))
       (setr obj
             (build rel  (build lex (lisp-escape (getr |*|)))
                    arg1 (lisp-escape (getr obj))))
       (to rationale-clause-adjuncts))
  (jump rationale-clause-adjuncts t))
```

```
(rationale-clause-adjuncts
  (push pp t
        (sendr v v)
        (addr vmods |*|)
        (to rationale-clause-adjuncts))
  (jump rationale-clause-end t))


(rationale-clause-end
  (pop (progn (setr rc
                    (lisp-escape
                      (eval
                        '(build agent  (lisp-escape (getr subj))
                                verb   (build lex (lisp-escape (getr v)))
                                object (lisp-escape (getr obj))
                                . ,(lisp-escape (getr vmods))))))
              (new-case-frame :rationale-clause 'rc)
              (liftr case-frames)
              (getr rc))
       t))


(NP
  (cat det t (setr art |*|) (to np-art))
  (jump np-art t))


(np-art
  (cat quant t (setr quant |*|) (to np-quant))
  (jump np-quant t))


(np-quant
  (cat adj t
       (addl adjs |*|
             (if (get-lexical-feature 'pos (getr features)) :pos :adj))
       (to np-quant))
  (jump np-adj t))


(np-adj
  (cat npr t    ; real noun
       (setr nh
             (find named-
```

```
                (lisp-escape
                  (or (find (sneps::compose sneps::name sneps::lex)
                            (lisp-escape (getr |*|)))
                      (build named
                             (lisp-escape (new-base-node dummy))
                             name
                             (build lex (lisp-escape (getr |*|)))))))))
      (lisp:assert (atom (getr nh)) (getr nh)
                   "NH should be a single node, not a list: ~s" (getr nh))
      (case (get-lexical-feature 'gen (getr features))
        (m  (sneps::=-fun (getr nh) 'most-recent-male))
        (f  (sneps::=-fun (getr nh) 'most-recent-female))
        (T  nil))
      (setr n |*|)
      (setr n-type :proper-noun)
      (setr nu (get-lexical-feature 'num (getr features)))
      (to np-n))
(cat pron t
      (setr nh (case (get-lexical-feature 'gen (getr features))
                 (m  (node-named most-recent-male))
                 (f  (node-named most-recent-female))
                 (n  (error "We don't deal with 'it'"))
                 (t  (error "Unknown gender feature"))))
      (lisp:assert (not (null (getr nh))) ()
                   "Couldn't find reference for pronoun.  ~
             Have we encountered a proper noun of the right gender yet?")
      (setr n |*|)
      (setr n-type :pronoun)
      (to np-n))
(cat n t                                          ; real noun
      (setr n |*|)
      (setr n-type :common-noun)
      (setr noun-class (lisp-escape (class-name (getr |*|))))
      (if (equal (getr art) "the")
          (setr nh
                (lisp-escape
                  (eval `(let ((node
                                (find (sneps::member- sneps::class)
                                      ,(getr noun-class))))
                           (lisp:assert
                             (null (cdr node)) (node)
                             "'The X' should yield only a single node, ~
```

```
                                not many: ~s" node)
                              (if node
                                  (progn (eval '(assert rel     definite
                                                        defarg  ,node))
                                         node)
                                  (progn
                                    (assert class  ,(getr noun-class)
                                            member (new-base-node dummy))
                                    (assert rel    definite
                                            defarg   (node-named dummy))
                                    (node-named dummy)))))))
            (setr nh (lisp-escape
                       (eval '(progn (assert class  ,(getr noun-class)
                                             member (new-base-node dummy))
                                     (assert rel    indefinite
                                             defarg   (node-named dummy))
                                     (node-named dummy))))))
          (case (get-lexical-feature 'gen (getr features))
            (m  (sneps::=-fun (getr nh) most-recent-male))
            (f  (sneps::=-fun (getr nh) most-recent-female))
            (T  nil))
          (setr nu (get-lexical-feature 'num (getr features)))
          (to np-n))
    (cat n nil                                          ; for complex noun?
          (setr n |*|)
          (addl adjs |*| :noun)
          (to np-adj)))

;; In certain circumstances, proper nouns and pronouns can take PP
;; modifiers (e.g., ''Tom in the next room'' or ''You with the nose,'')
;; but allowing this usually means that the first parse is an unwanted
;; parse, so we are disallowing PP modification of nouns other than
;; common nouns.
;; SDA 5/10/89

{\tt \noindent
(np-n
  (push pp ;; hack alert!
        (and
          (eq (getr n-type) :common-noun)
          ;; either both can apply to a physical object, or neither (hack!)
          (eq (get-lexical-feature 'non-physical (getr features))
              (get-lexical-feature 'non-physical (getr n))))
```

```
        (sendr v v)
        (build arg1 (lisp-escape (getr nh))
               rel  (lisp-escape (car (getr |*|)))
               arg2 (lisp-escape (cadr (getr |*|))))
        (to np-n))
  (pop  (progn
          (lisp:assert (not (null (getr nh))) () "register NH is null")
          (do* ((adjs (getr adjs) (cddr adjs))
                (word (car adjs) (car adjs))
                (prop (cadr adjs) (cadr adjs)))
               ((null adjs))
            ;; because of odd behavior by ADDL
            (eval `(assert arg1 ,(getr nh)
                           rel  (build lex ,(case prop
                                              ((:adj :noun) 'has-property)
                                              (:pos 'is-part)))
                           arg2 (build lex ,word))))
          (getr nh))
        ;; adjs - adjectives -- associate via hasproperty-link
        ;; the predicate DETAGREE tests for agreement between
        ;; the ART and N registers to screen out ``a books''
        ;; and ``an table''
        (detagree)))

(pp
  (cat prep (not (member (getr |*|) (getf :particles (getr v))
                         :test #'string-equal))
       ;; This code converts the prep to a relation
       (setr tmp |*|)
       (if (not (stringp (getr tmp)))
           (error "preposition is not a string:  ~s" (getr tmp)))
       (setr tmp (string-upcase (getr tmp)))
       (setr tmp (intern (getr tmp) (find-package 'sneps)))
       (if (not (relation? (getr tmp)))
           (error "preposition is not a relation:  ~s" (getr tmp)))
       (setr prep tmp)
       (to pp-prep)))

(pp-prep
  (push np (member (getf ctgy) '(n npr det adj))
        ;; predicate fails if the next word cannot begin an NP
        (sendr v v)
```

```
        (setr np |*|)
        (to pp-np)))

(pp-np (pop (list (getr prep) (getr np)) t))
```

NAIC Final Report: Discussing, Using and Recognizing Plans

No-cost Extension, September–November, 1989

Beverly Woolf
Computer & Information Science
University of Massachusetts
Amherst, MA 01003

During the period September-November, 1989, the University of Massachusetts group adapted their system to use SNePS 2.1, and successfully completed tests parsing the tutoring paragraph and questions relevant to the tutoring session. The tasks carried out during this period include:

1. adaptation to SNePS 2.1
2. refinements to rules for answering questions
3. modifications to focussing to make it work exactly as advertised
4. refinements to representations generated by parser
5. merging of knowledge bases used to support answers to different questions.

Two problems interfered with completion of the work:

1. Massive amounts of trace output: During a test run, up to 12000 lines of output could be generated, most of which concerns multiple deductions producing identical inferences. Wading through all the output is almost impossible, making the task of off-line debugging very difficult.
2. Non-intuitive nature of trace output: It was very difficult to follow a chain of reasoning, since the trace output is ordered according to when inferences

are made. Consequently, the trace output was virtually useless to us, beyond serving as proof that a deduction had been successfully made. The introduction of a facility which shows the chain of reasoning would be invaluable.

These are not intended as criticisms of the SNePS system but rather as feedback from users on how the system may be made more convenient to use.

Question 3: The UMass system is designed to answer three questions about a paragraph that was parsed and comprehended by the system (see Final NAIC report). Until very recently, we thought we had a solution to Question 3, but we now realize that answering this question is much more complicated than we had thought. While we no longer have time to devise and program a complete revised solution, we can sketch out our thoughts on the problem.

Essentially, to answer the question

"Why did Nancy ask Tom if inanimate objects exert a force?"

the system must understand that Tom's knowing the domain of Statics is sufficient for him to answer Nancy's question, and therefore the question is a good one for probing his knowledge of the domain. We would like to be able to say that is it *necessary* and sufficient that one know a domain in order to answer a question about it. However, this is clearly not always the case; for example, Tom could cheat.

For the system simply to know that the question is about Statics is difficult, because it is not enough to know that each term in the question is in the domain, since the terms may be related in such a way that the composite meaning falls outside the domain. Now, to some extent we can control which relations appear in the composite meaning representation, via our control of the ATN, but this would be an unsatisfactory *ad hoc* solution.

Because a realistic depth of understanding in the knowledge representation is so difficult to achieve, we have simply chosen to leave this question for possible subsequent studies.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic reliability/maintainability and compatibility.